

Lucerne University of  
Applied Sciences and Arts

**HOCHSCHULE  
LUZERN**

Technik & Architektur  
CC Innovation in Intelligent  
Multimedia Sensor Networks

## Benutzer Dokumentation für Raspberry Pi

---

Document Status: on-going  
Issue Date: 24.05.2018  
Author(s): Klaus Zahn

### Document History

<b>Revision</b>	<b>Date</b>	<b>Changes</b>	<b>Author</b>
0.9	28.03.2018	Document Setup	Klaus Zahn
1.0	22.05.2018	Added Übung 9	Klaus Zahn

# Contents

<b>1. Einleitung .....</b>	<b>4</b>
<b>2. Inbetriebnahme .....</b>	<b>5</b>
2.1. Das Raspberry Pi Kit und Zusatz SW .....	5
2.2. Allgemeiner Überblick .....	5
2.3. Installation .....	6
2.3.1. Anbindung des Raspberry Pi per Ethernet an den Host.....	6
2.3.2. Kommunikation mit dem Raspberry Pi via Browser .....	8
2.3.3. Kommunikation mit dem Raspberry Pi via Putty.....	9
2.3.4. Kommunikation mit dem Raspberry Pi via VNC.....	10
<b>3. Installation der VM mit dem SDK.....</b>	<b>12</b>
3.1. Allgemeine Bemerkungen zur Verwendung der Ubuntu VM.....	13
3.1.1. Eine Liste von einigen nützlichen Befehlen.....	14
3.2. Kommunikation zwischen Host und VM.....	14
3.2.1. Test der Netzwerk Interfaces der VM.....	14
3.2.2. Datenaustausch via „Gemeinsamer Ordner“ .....	16
3.2.3. Kommunikation von Host zu VM via SSH.....	17
<b>4. Verwendung des Raspberry Pi SDK .....</b>	<b>18</b>
4.1. Ein erstes Hello World Programm .....	18
4.1.1. Übung 1: Entwicklung eines „Hello World“ Programms für Linux in C.....	18
4.1.2. Übung 2: Entwicklung eines „Hello World“ Programms für Raspbian in C .....	20
4.1.3. Übung 3: Entwicklung eines „Hello World“ in C++ .....	21
4.1.4. Übung 4: Erstellung und Einbindung einer Library/Skripting .....	22
4.2. Anwendungsbeispiele für OpenCV Library .....	26
4.2.1. Übung 5: Einbindung der OpenCV Library .....	26
4.2.2. Übung 6: Verwendung des Boa Webservers .....	28
4.2.3. Übung 7: Applikation Template .....	32
<b>5. Eine Einführung in Makefiles .....</b>	<b>41</b>
5.1. Quick Reference .....	41
5.2. Erweiterte Einführung in Makefiles .....	41
5.3. Targets, Regeln und Abhängigkeiten.....	42
5.4. Das Default Target.....	43
5.5. Pattern in Regeln .....	43
5.6. Variablen in Makefiles .....	44
5.7. Kommentare in Makefiles .....	44
5.8. Phony targets .....	44

5.9. Pattern Substitution.....	45
5.10. Abhängigkeiten als Target (make dep) .....	45
5.11. Rekursives Make .....	46
<b>6. Anwendungen.....</b>	<b>48</b>
6.1. Übung 8: Change Detection.....	48
6.2. Übung 9: OCR .....	54
<b>7. Details zu VirtualBox .....</b>	<b>60</b>
7.1. Netzwerk Einstellungen .....	60
<b>8. Raspberry Pi .....</b>	<b>62</b>
8.1. Starten der Default Applikation.....	62
8.2. Verbindung ohne Passwort Eingabe .....	62
<b>9. Annex .....</b>	<b>64</b>
9.1. References.....	64
9.2. Abkürzungen .....	64
9.3. git in a Nutshell.....	64
9.3.1. Verwendung von github.....	66
9.4. Trouble Shooting.....	68
9.4.1. Manual Page .....	68
9.4.2. Issues bei der SSH-Verbindung mit dem Raspberry Pi .....	68
9.4.3. Raspberry Pi Kamera sendet keine Bilder .....	69
9.4.4. Gemeinsamer Ordner nicht verfügbar.....	70
9.4.5. Keine Schreibrechte auf dem Gemeinsamen Ordner .....	70
9.4.6. Die Alt Taste funktioniert in der VM nicht .....	71
9.4.7. USB Stick mit der VM verbinden .....	71
9.4.8. Falsche Namen der Netzwerk Adapter der VM (nicht eth0 und eth1) .....	72
9.4.9. Konflikt unter Eclipse IDE beim Öffnen eines Projekts.....	72
9.4.10. Fehler beim Mounten des Gemeinsamen Ordners.....	73
9.5. Erstellung der VM “from scratch” .....	74

## 1. Einleitung

Die Einleitung der Wikipedia Seite zum Raspberry Pi fasst die Philosophie der Raspberry Pi Hardware (HW) gut zusammen [1]:

*„Der Raspberry Pi ist ein Einplatinencomputer, der von der gemeinnützigen britischen Raspberry Pi Foundation entwickelt wurde. Der Rechner enthält ein Ein-Chip-System von Broadcom mit einem ARM-Mikroprozessor, die Grundfläche der Platine entspricht etwa den Abmessungen einer Kreditkarte. Der Raspberry Pi kam Anfang 2012 auf den Markt; sein großer Markterfolg wird teils als Revival des bis dahin weitgehend bedeutungslos gewordenen Heimcomputers zum Programmieren und Experimentieren angesehen. Der im Vergleich zu üblichen Personal Computern sehr einfach aufgebaute Rechner wurde von der Stiftung mit dem Ziel entwickelt, jungen Menschen den Erwerb von Programmier- und Hardwarekenntnissen zu erleichtern. Entsprechend niedrig wurde der Verkaufspreis angesetzt, der je nach Modell etwa 5 bis 35 USD beträgt.“*

In Abbildung 1 ist die Plattform (Version Raspberry Pi 2) mit Kamera und USB WLAN Adapter dargestellt.



**Abbildung 1:** Die Raspberry Pi HW.

Wesentliches Ziel der Verwendung der Plattform im Modul EBV ist, einfache Anwendungen der Bildverarbeitung auf einem echtzeitfähigen Zielsystem zu implementieren. Zusätzlich bietet die Verwendung der Raspberry Pi Plattform noch die Möglichkeit, einen einfachen Zugang zu eingebetteten Systemen („embedded Systems“) und dem in der Regel hierfür verwendeten Betriebssystem Linux zu ermöglichen. Dabei ist die verwendete ARM-Architektur [2] eine der meist genutzten im embedded Bereich, inklusive Smartphones.

## 2. Inbetriebnahme

### 2.1. Das Raspberry Pi Kit und Zusatz SW

Das Raspberry Pi Kit besteht aus:

1. Raspberry Pi HW mit Kamera
2. USB-Kabel zur Stromversorgung
3. Ethernet Kabel
4. Micro SD Karte mit dem Betriebssystem
5. SD Adapter für Micro SD
6. USB WLAN Adapter



Abbildung 2: *Raspberry Pi Kit.*

Weiterhin finden sich auf ILIAS im Verzeichnis SW noch verschiedene Installationspakete, auf die wir im Folgenden verweisen werden.

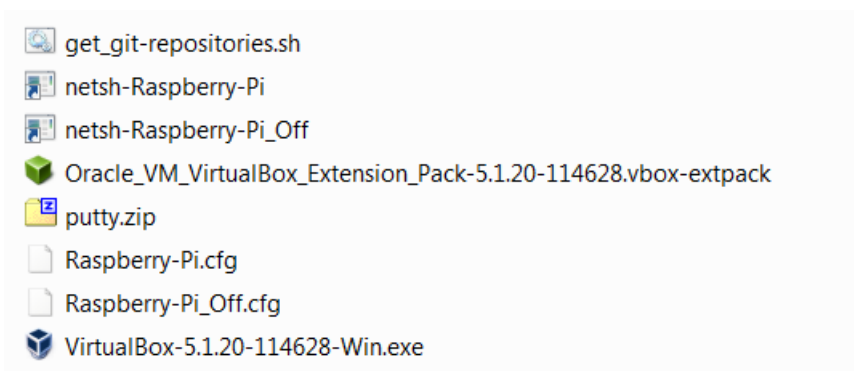


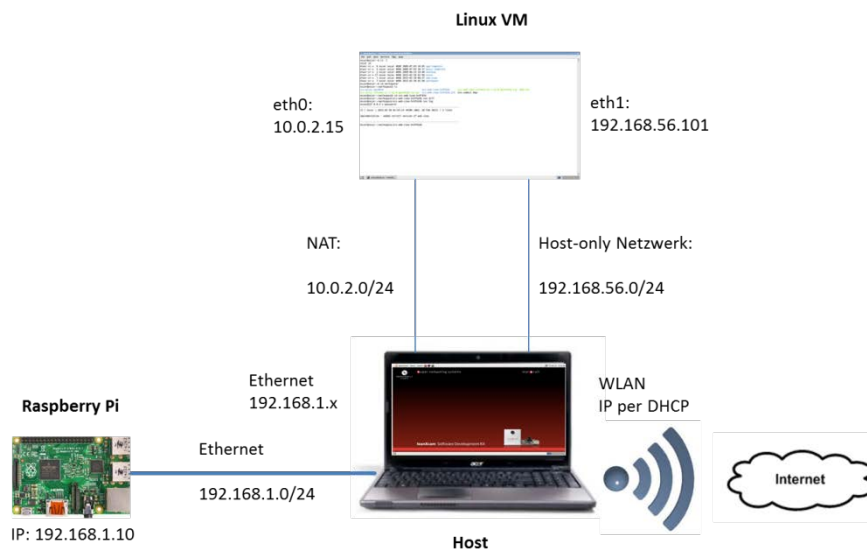
Abbildung 3: *Verzeichnis Raspberry-Pi \ SW auf ILIAS*

### 2.2. Allgemeiner Überblick

Die Verteilung (und auch die Verwendung) des Software Development Kits (SDK), d.h. der für die Programmierung auf der Raspberry Pi HW notwendigen SW-Pakete, erfolgt

über eine virtuelle Maschine (VM)<sup>1</sup>. Dies hat den Vorteil, dass die nicht immer einfache Installation der Programme inklusive der verschiedenen Abhängigkeiten schon vorab durchgeführt werden können. Die VM wird über die Virtualisierungssoftware VirtualBox verwaltet. In der folgenden Abbildung 4 sind die involvierten HW- und SW-Komponenten im Überblick dargestellt:

- **Host:**  
PC oder Laptop mit einem Ethernet Interface, zur Anbindung der Raspberry Pi-HW sowie einem WLAN Interface zum Internet Access.
- **Raspberry Pi:**  
eingebettete Plattform mit Kamera für Bildverarbeitung.
- **Linux VM:**  
Virtuelle Maschine, welche das SDK für die Entwicklung für die Raspberry Pi Plattform beherbergt.



**Abbildung 4:** Überblick über die HW- und SW-Komponenten und deren gegenseitige Anbindung.

In Abbildung 4 sind auch die verschiedenen Netzwerkkomponenten eingezeichnet, die zur Kommunikation zwischen den verschiedenen Komponenten notwendig sind<sup>2</sup>. Details zur Konfiguration finden sich in Abschnitt 7.1. Ziel bei dieser Konfiguration<sup>3</sup> ist, dass eine bidirektionale Kommunikation zwischen Host und VM besteht, als auch, dass von der VM ebenso wie vom Host direkt auf das Raspberry Pi zugegriffen werden kann.

## 2.3. Installation

Die Installation gliedert sich in verschiedene Schritte, die im Folgenden beschrieben werden sollen

### 2.3.1. Anbindung des Raspberry Pi per Ethernet an den Host

Die Raspberry Pi Kits werden mit der IP 192.168.1.10/24<sup>4</sup> ausgeliefert. Die physikalische Anbindung erfolgt über den Ethernet Adapter des Host mit Hilfe des im

<sup>1</sup> Die virtuelle Maschine ist eine Ubuntu Linux Distribution (Version 16.04, LTS).

<sup>2</sup> Zum Fileaustausch zwischen Host und VM existiert auch die Option eines „Gemeinsamen Ordners“.

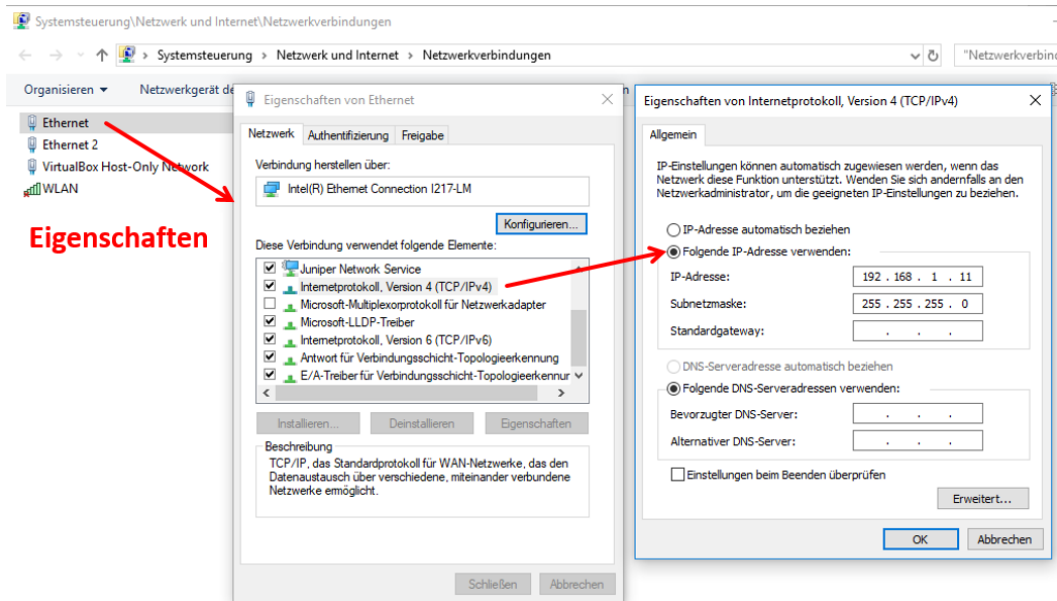
<sup>3</sup> Die Default Konfiguration einer VM auf der VirtualBox besteht nur aus dem NAT Interface, ohne die Möglichkeit eines Zugriffs vom Host auf die VM.

<sup>4</sup> Für die Bedeutung dieser Schreibweise siehe [3].

Raspberry Pi Kit enthaltenen Ethernet-Kabels (Abbildung 2, Item 3). Für eine funktionierende IP-Kommunikation zwischen Host und Raspberry Pi muss die Ethernet Adresse des Host auf einen Wert im Subnetz 192.168.1.0/24 gesetzt werden. D.h. alle IP Adressen angefangen von 192.168.1.1 bis 192.168.1.254<sup>5</sup> (mit Ausnahme von 192.168.1.10, welche vom Raspberry Pi besetzt ist) können verwendet werden. Unter Windows<sup>6</sup> können die Netzwerkeinstellungen unter

Systemsteuerung\Netzwerk und Internet\Netzwerkverbindungen

gemäss Abbildung 5 vorgenommen werden.



**Abbildung 5:** Beispiel für das Setzen der IP-Adresse 192.168.1.11/24.

Das manuelle Setzen der IP Adresse über die Systemsteuerung von Windows ist auf Dauer mühsam. Eine einfache Möglichkeit, die Netzwerk Adapter Einstellungen per Skript zu ändern bietet das Programm `netsh.exe`, das im Umfang des Windowsbetriebssystems standardmässig mitgeliefert wird. Dies wird mit Hilfe der Skripte `netsh-Raspberry-Pi` und `netsh-Raspberry-Pi_Off` im SW-Verzeichnis (Abbildung 3) realisiert. Wenn beide Skripte – zusammen mit den zugehörigen Konfigurationsdateien `Raspberry-Pi.cfg` bzw. `Raspberry-Pi_Off.cfg` – in das Verzeichnis `C:\Raspberry-Pi` kopiert werden, kann durch ausführen des Skripts `netsh-Raspberry-Pi` (als Administrator unter Windows) die IP Adresse des Ethernet Interfaces<sup>7</sup> auf den Wert 192.168.1.11/24 gesetzt werden. Ausführen des Skripts `netsh-Raspberry-Pi_Off` (als Administrator unter Windows) setzt den Wert wieder auf eine automatische IP-Konfiguration per DHCP zurück.

Für einen ersten Test zur Verifikation der Ethernet Verbindung mit dem Raspberry Pi können Sie auf der Kommandozeile einen ping-Request schicken (Abbildung 6). Sollte der Raspberry Pi bei korrekter Einstellung des Netzwerkes nicht antworten, so hilft in

<sup>5</sup> Die Adresse 192.168.1.255 ist die Broadcast Adresse des Subnetzes 192.168.1.0/24 und kann daher – ebenso wie die Netzwerkadresse 192.168.1.0 selbst – nicht verwendet werden.

<sup>6</sup> Für die Betriebssysteme Linux bzw. Mac OS verwenden Sie ggf. die zugehörigen „Hilfe und Support“ Anweisungen.

<sup>7</sup> Ggf. müssen Sie den Namen des Interfaces (in den Konfigurationsfiles unter `name="LAN-Verbindung"`) anpassen. Hierzu können Sie mit dem Befehl `ipconfig` auf der Kommandozeile die Bezeichnung des Ethernet Interfaces herausfinden.



der Regel ein Neustart des Raspberry Pi (bei bestehender Anbindung per Ethernet Kabel)<sup>8</sup>.

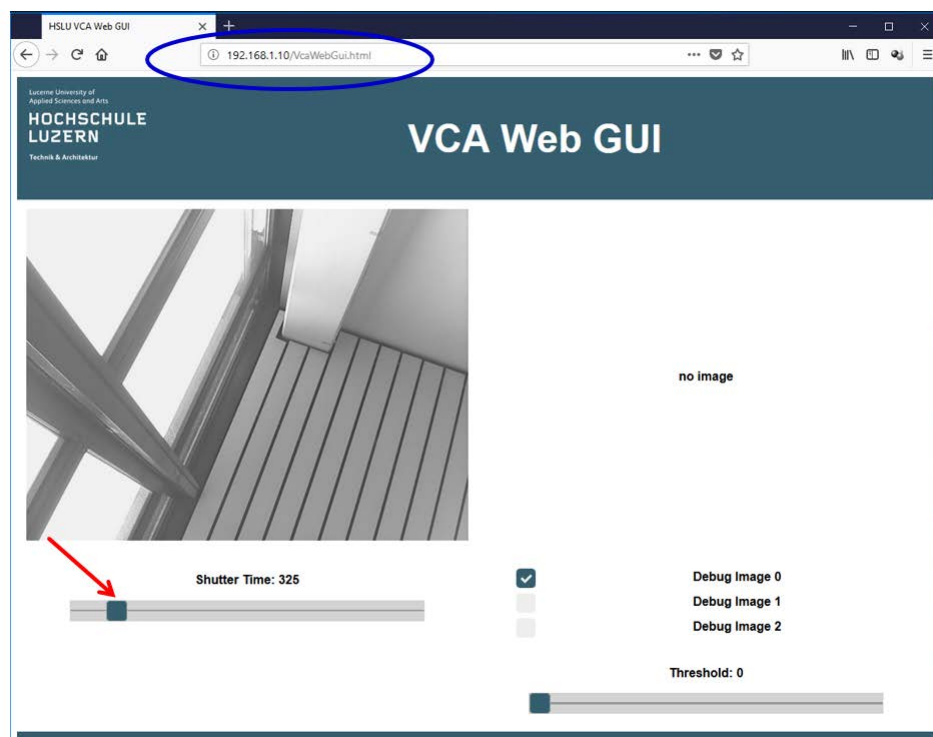
```
c:\>ping -t 192.168.1.10

Ping wird ausgeführt für 192.168.1.10 mit 32 Bytes Daten:
Antwort von 192.168.1.10: Bytes=32 Zeit<1ms TTL=64
Antwort von 192.168.1.10: Bytes=32 Zeit<1ms TTL=64
Antwort von 192.168.1.10: Bytes=32 Zeit<1ms TTL=64
Antwort von 192.168.1.10: Bytes=32 Zeit<1ms TTL=64
Antwort von 192.168.1.10: Bytes=32 Zeit<1ms TTL=64
Antwort von 192.168.1.10: Bytes=32 Zeit<1ms TTL=64
Antwort von 192.168.1.10: Bytes=32 Zeit<1ms TTL=64
```

**Abbildung 6:** Test der Kommunikation zwischen Host und Raspberry Pi mittels Ping.

### 2.3.2. Kommunikation mit dem Raspberry Pi via Browser

Auf den Raspberry Pi Kits ist standardmässig die Applikation `vcagrabber` installiert, die vom installierten Boa [4] Webserver gesteuert wird. Die Kommunikation vom Host findet via Browser statt. Hierzu einen Webbrowser öffnen, in der Adresszeile die IP-Adresse des Raspberry Pi (192.168.1.10) eingeben (Abbildung 7) und die Enter Taste drücken. Es öffnet sich eine Webseite, auf der verschiedene Einstellungen vorgenommen werden können. Diese werden wir im weiteren Verlauf noch im Detail kennen- und verwenden lernen. Interessant ist aktuell lediglich die Einstellung der «Shutter Time» (d.h. die Belichtungszeit) (Pfeil). Für den Wert „0“ (Slider ganz links) wird die Belichtungszeit automatisch gewählt („Auto Exposure“) und für Werte grösser Null wird eine fixe Einstellung verwendet.



**Abbildung 7:** Kommunikation via Browser mit dem Raspberry Pi.

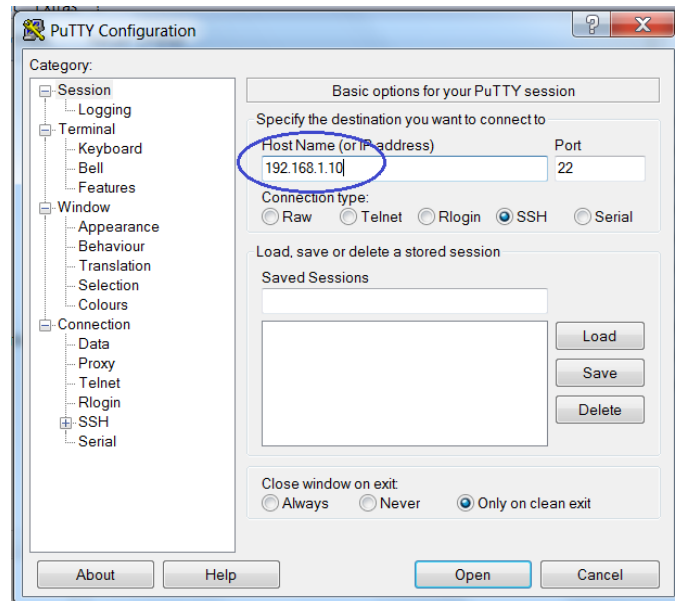
<sup>8</sup> Die folgende Reihenfolge, bei der Installation scheint am robustesten: 1) IP-Adresse des Ethernet Netzwerkinterfaces des Host setzen -> 2) Ethernet Kabel verbinden -> 3) Netzkabel des Raspberry Pi verbinden und so den Raspberry Pi starten.



### 2.3.3. Kommunikation mit dem Raspberry Pi via Putty

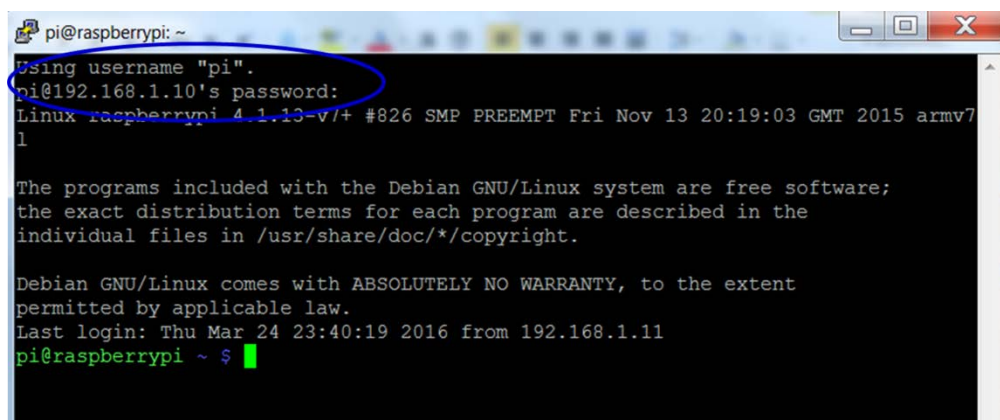
Putty ist ein Client mit dem u.a. eine Secure Shell (SSH) aufgebaut werden kann. SSH ist (neben Telnet) das Standardprotokoll für eine sichere Remoteverbindung mit einem Linux Rechner. Daher eignet sich Putty für die Kommunikation mit dem Raspberry Pi. Das Programm Putty befindet sich im SW Verzeichnis auf ILIAS (Abbildung 3). Dieses Programm muss nicht installiert werden, sondern kann an einem beliebigen Ort ausgepackt und verwendet werden.

Starten Sie das Programm putty.exe im putty Verzeichnis und geben Sie im Dialog im Feld „Host Name“ die IP-Adresse des Raspberry Pi ein (Abbildung 8). Drücken Sie dann auf die Taste „Open“.



**Abbildung 8:** Dialog der putty Anwendung zur Öffnung einer SSH-Verbindung.

Danach öffnet sich ein Terminal Fenster (Abbildung 9) mit der Aufforderung zur Eingabe des Benutzeramens `pi` und des Passwords `hslu` (beides in Kleinbuchstaben). Damit steht die SSH-Verbindung zum Linux Betriebssystem des Raspberry Pi zur Verfügung.



**Abbildung 9:** Terminal Fenster einer SSH-Verbindung via Putty zur Raspberry Pi.

```

pi@raspberrypi: ~/tmp/vcagrabber
top - 17:40:47 up 9 min, 2 users, load average: 0.39, 0.44, 0.29
Tasks: 116 total, 1 running, 78 sleeping, 0 stopped, 0 zombie
%Cpu(s): 9.9 us, 0.5 sy, 0.0 ni, 89.2 id, 0.0 wa, 0.0 hi, 0.4 si, 0.0 st
KiB Mem : 896704 total, 608656 free, 88320 used, 199728 buff/cache
KiB Swap: 102396 total, 102396 free, 0 used, 746044 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 1014 root        20   0   18564   7164   6188  S   37.4   0.8   0:46.42 vcagrabberd
 1050 pi          20   0    8104    3180   2724  R    1.3   0.4   0:00.14 top
   59 root        1 -19     0     0     0  S   1.0   0.0   0:04.32 vchiq-slot/0
  682 pi          20   0  139776  23808  19916  S    0.7   2.7   0:04.31 lxpanel
   25 root        20   0     0     0     0  I   0.3   0.0   0:00.34 kworker/3:0
  519 root        20   0  124896  34520  23528  S    0.3   3.8   0:02.30 Xorg
  853 root        20   0     0     0     0  I   0.3   0.0   0:00.18 kworker/0:1
    1 root        20   0    9736    6156   4940  S    0.0   0.7   0:03.82 systemd
    2 root        20   0     0     0     0  S    0.0   0.0   0:00.00 kthreadd
    3 root        20   0     0     0     0  I   0.0   0.0   0:00.12 kworker/0:0
    4 root         0 -20     0     0     0  I   0.0   0.0   0:00.00 kworker/0:0H
    6 root         0 -20     0     0     0  I   0.0   0.0   0:00.00 mm_percpu_wq
    7 root        20   0     0     0     0  S    0.0   0.0   0:00.13 ksoftirqd/0
    8 root        20   0     0     0     0  I   0.0   0.0   0:00.46 rcu_sched
    9 root        20   0     0     0     0  I   0.0   0.0   0:00.00 rcu_bh
   10 root        rt    0     0     0     0  S    0.0   0.0   0:00.00 migration/0
   11 root        20   0     0     0     0  S    0.0   0.0   0:00.00 cpuhp/0

```

**Abbildung 10:** Der Linux Befehl `top` zeigt die laufenden Prozesse an.

Geben Sie als Test auf der Kommandozeile den Befehl `top` ein. Dann werden alle aktuell laufenden Prozesse angezeigt (Abbildung 10). Z.B. (roter Pfeil) ist die laufende `vcagrabber` Applikation zu erkennen, sowie `top` in der Zeile darunter.

Verlassen Sie die Anzeige des `top`-Befehls mit `Ctrl-C` und schliessen Sie die SSH-Verbindung mittels `exit`.

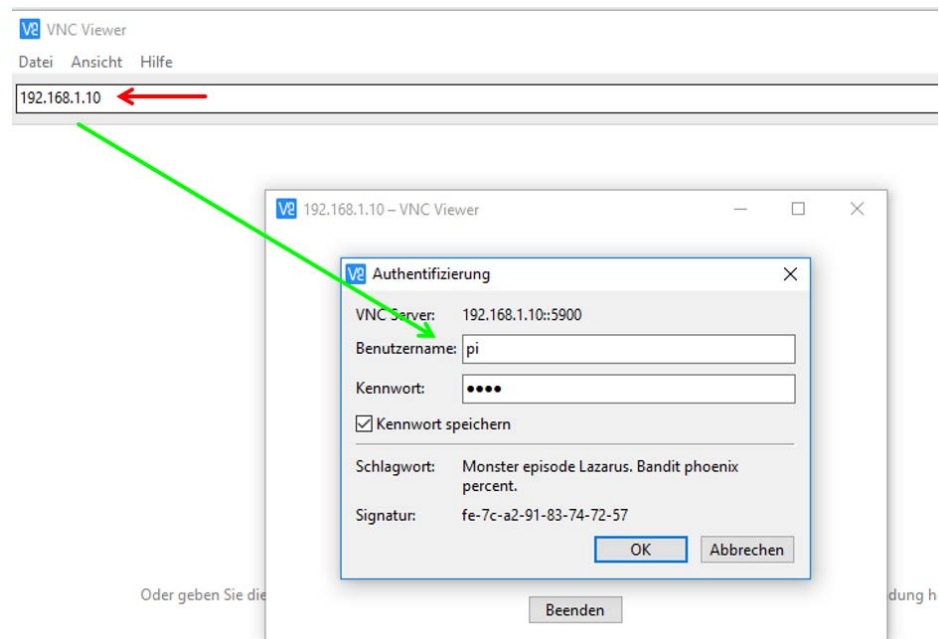
### 2.3.4. Kommunikation mit dem Raspberry Pi via VNC

VNC (Virtual Network Computing) ist ein SW-Tool, um einen Rechner (VNC-Server) via Netzwerkverbindung von einem (VNC-)Client remote zu steuern. Der VNC-Server ist bereits auf dem Raspberry Pi installiert, so dass Sie lokal auf Ihrem Rechner nur den VNC-Client (VNC-Viewer) installieren müssen. Diesen können Sie sich von der entsprechenden Webseite herunterladen<sup>9</sup>.

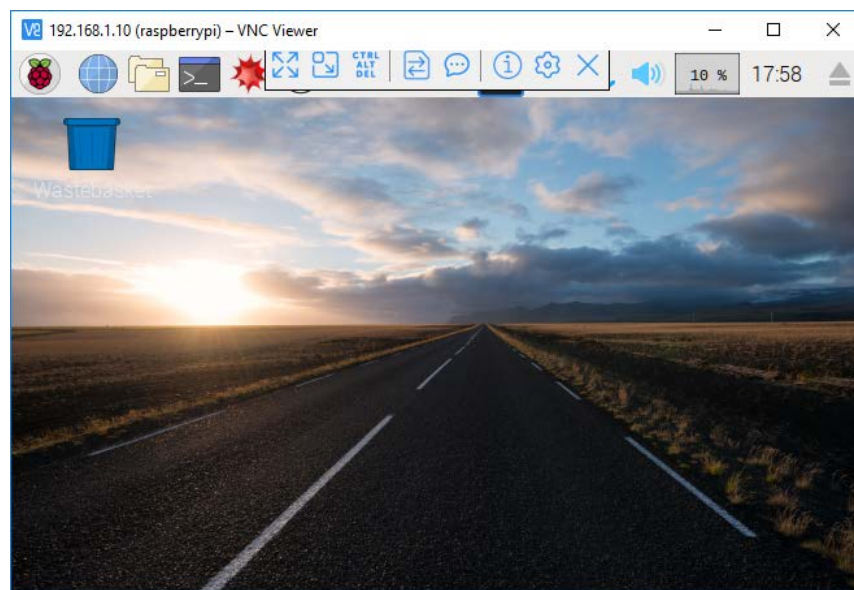
Nach der Installation öffnen Sie den VNC-Viewer (Abbildung 11), geben die IP-Adresse des Raspberry Pi ein (roter Pfeil) und drücken die Eingabe Taste. Quittieren Sie etwaige Warnungen, geben den Benutzernamen `pi` und das Passwort `hslu` (grüner Pfeil) und Drücken auf die Ok Taste. Dann wird ein Fenster geöffnet und es erscheint das der Desktop des Raspberry Pi, wie es sehen würden, wenn Sie einen Monitor an den HDMI-Ausgang des Raspberry Pi anschliessen würden (Abbildung 12). Dies zeigt, dass der Raspberry Pi ein vollständiger Computer ist und prinzipiell als Desktop Ersatz verwendet werden könnte. So wäre es z.B. möglich, die Applikationen zur Bildverarbeitung, welche wir entwickeln werden, direkt auf dem Raspberry Pi zu erstellen und zu kompilieren. Allerdings sind die Ressourcen hierfür etwas limitiert.

Im Modul EBV werden wir den Raspberry Pi allerdings wie eine eingebettete Plattform verwenden und die Applikation – per Cross Compiler – auf dem Host einwickeln, dann auf den Raspberry Pi übertragen und remote das Ergebnis analysieren.

<sup>9</sup> <https://www.realvnc.com/de/connect/download/viewer/>



**Abbildung 11:** Verbindung mittels VNC-Viewer zum Raspberry Pi.



**Abbildung 12:** Desktop des Raspberry Pi im VNC Viewer.

**Hinweis:** An dieser Stelle haben wir die grundsätzliche Funktionalität des Raspberry Pi verifiziert und die für einen *Anwender* notwendigen Kommunikationsschnittstellen bereitgestellt. Nun wenden wir uns der Installation der Programmierungsumgebung für die *Entwicklung* auf dem Raspberry Pi zu.

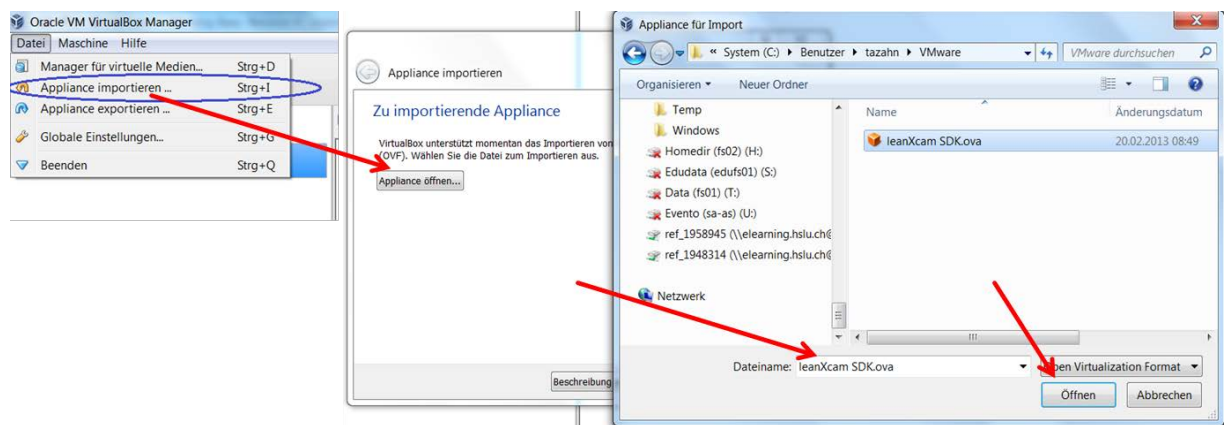
### 3. Installation der VM mit dem SDK

Wie einleitend bereits bemerkt, wird das Software Entwicklungskit (SDK) des Raspberry Pi über eine Virtuelle Maschine (VM), welches eine Ubuntu Linux Distribution (Version 16.04, LTS) enthält, ausgeliefert. Um die VM zu starten wird die Virtualisierungssoftware VirtualBox benötigt. Das Installationspaket für Windows<sup>10</sup> liegt im Raspberry-Pi\SW Verzeichnis auf ILIAS (Abbildung 3).

Nach der Installation der VirtualBox muss die VM installiert werden. Verwenden Sie hierzu die im Unterricht per USB-Stick verteilte VM. Letztere beinhaltet schon verschiedene Updates.

Gehen Sie dazu wie in Abbildung 13 gezeigt vor:

1. Appliance importieren auswählen
2. nach dem Klick auf Appliance öffnen
3. im nachfolgenden Filedialog das ova-File auf dem Memory Stick auswählen
4. dann importieren.



**Abbildung 13:** Importieren der Virtuellen Maschine mit dem Raspberry Pi-SDK in VirtualBox.

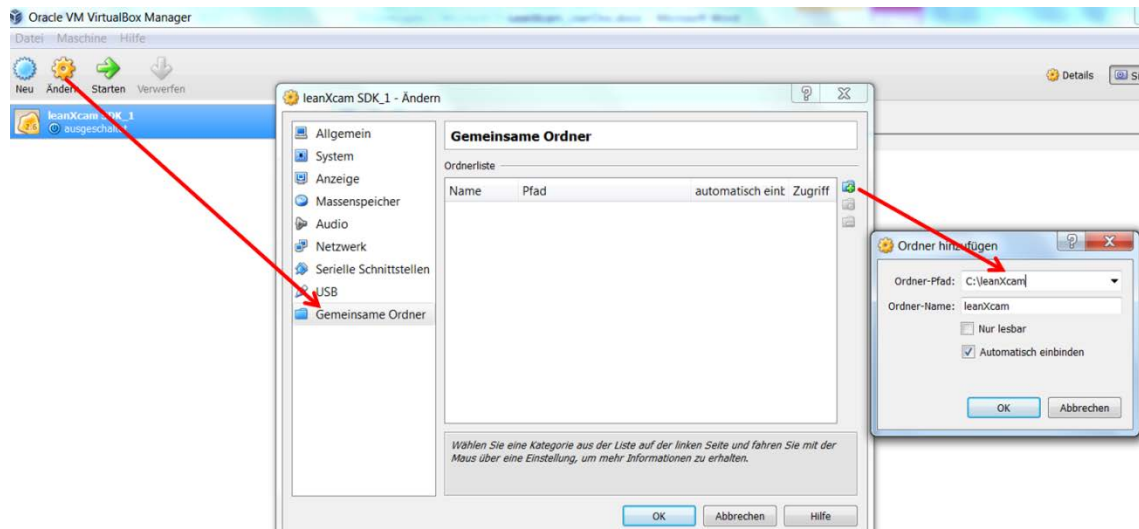
Bevor wir die VM starten, definieren wir noch einen „Gemeinsamen Ordner“, der einen einfachen Datenaustausch zwischen VM und Host erlaubt<sup>11</sup>. Dies ist in Abbildung 14 zusammengefasst:

1. Unter Ändern wählen Sie Gemeinsamer Ordner
2. dann im Dialog rechts auf das „Plus“ Symbol drücken
3. in dem darauffolgenden Dialog wählen Sie den Ordner im Host (d.h. Ihrem PC) aus, welchen Sie als Austauschordner verwenden wollen
4. wählen Sie ebenfalls Automatisch einbinden.

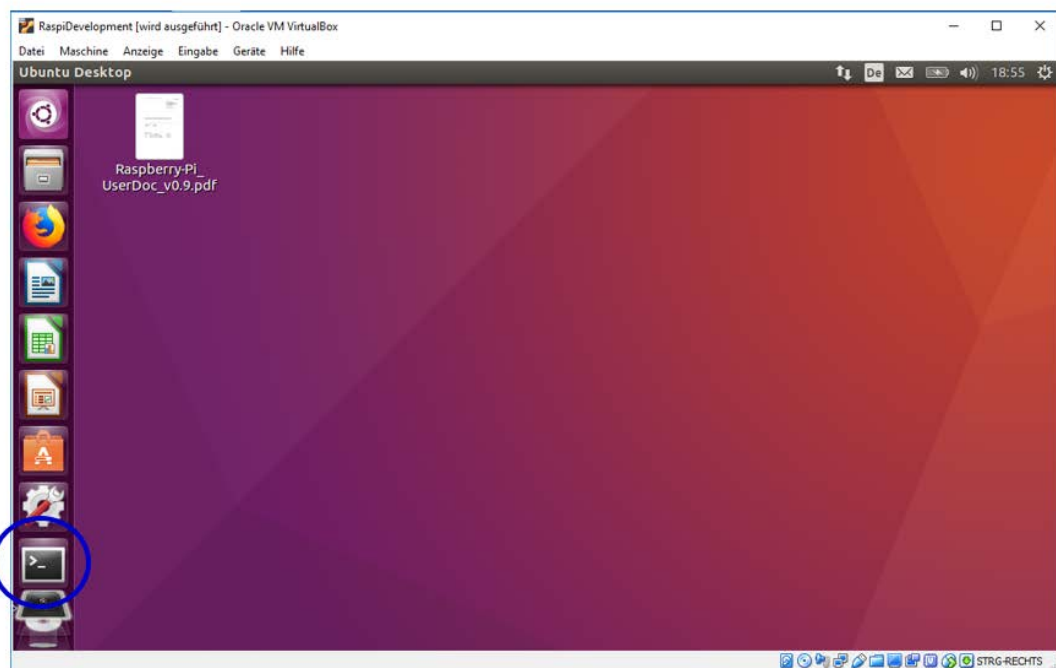
Dann können Sie auf den Pfeil drücken und damit die VM starten. Das Ubuntu Betriebssystem fährt dann hoch und der Default Desktop (Abbildung 15) wird sichtbar.

<sup>10</sup> Für Linux oder Mac finden Sie die Pakete unter [5].

<sup>11</sup> Ansonsten sind VM und Host nur über die Netzwerk Schnittstellen verbunden, was einen Fileaustausch etwas schwerfällig macht.



**Abbildung 14:** Definition eines Gemeinsamen Ordners zum einfachen Datenaustausch zwischen Host und VM.



**Abbildung 15:** Desktop der Ubuntu VM.

### 3.1. Allgemeine Bemerkungen zur Verwendung der Ubuntu VM

Die Ubuntu Distribution verfolgt das Ziel, eine möglichst einfache Verwendung von Linux zu ermöglichen. Daher kann über die graphische Benutzeroberfläche das gesamte System verwaltet werden. Allerdings empfiehlt es sich, die Verwendung der Kommandozeile kennenzulernen, da dessen Verwendung – nach einer gewissen Einübungszeit – wesentlich effizienter ist. Auch ist die Kenntnis dieselben Befehle für die Verwendung der SSH-Verbindung zur Raspberry Pi HW notwendig (Abschnitt



2.3.3). Ein Kommandozeilenfenster („Terminal“) wird durch Drücken der entsprechenden Taste auf dem Desktop (Abbildung 15, blauer Kreis) geöffnet.

### 3.1.1. Eine Liste von einigen nützlichen Befehlen

Viele der Befehle sind auch in ähnlicher Form im Kommandozeilenfenster von Windows zu verwenden. Mittels der Option `--help` können zusätzliche Informationen zu einem Befehl erhalten werden.

#### Filehandling:

- `ls -l`: Zeigt den detaillierten Inhalt des aktuellen Verzeichnisses an
- `pwd`: Zeigt den aktuellen Pfad an
- `cp`: Kopiert ein File oder ein Verzeichnis
- `rm`: löscht ein File oder ein Verzeichnis
- `chown -R pi:pi`: Ändert die Besitzer- und Gruppenzugehörigkeit der angegebenen Datein(en) rekursiv auf `pi`.

#### Prozesses:

- `chmod a+x app`: erteilt allen Benutzern Ausführrechte der Applikation `app`<sup>12</sup>
- `./app`: Startet die ausführbare Applikation `app`<sup>13</sup>
- `./run.sh`: Startet Shell Skript `run.sh`
- `top`: Zeigt alle aktuell laufenden Prozess an (mit `Ctrl-C` beenden)

#### Netzwerk:

- `ping 192.168.1.10`: Sendet einen Ping Request an den Raspberry Pi
- `ifconfig`: Zeigt bzw. ändert die aktuelle Netzwerkkonfiguration
- `sudo service network-manager restart`: Neustart d. Netzwerkinterfaces<sup>14</sup>
- `netstat -r`: Zeigt die Routing Tabelle an.
- `route del`: Löscht eine Route
- `route add`: Fügt eine Route hinzu

#### Makefiles:

- `make`: erstellt eine Applikation mittels Makefile
- `make clean`: löscht alle Binär- und Objektfiles einer Applikation

#### Remote:

- `ssh 192.168.1.10`: Stellt eine SSH-Verbindung zum Raspberry Pi her

## 3.2. Kommunikation zwischen Host und VM

### 3.2.1. Test der Netzwerk Interfaces der VM

Wie in Abbildung 4 dargestellt, verfügt die VM über zwei Netzwerk Interfaces<sup>15</sup>. Für einen ersten Test kann auf der Konsole eines Terminals der Befehl `ifconfig` ausgeführt werden. Es sollten – wie in Abbildung 16 gezeigt – die beiden Netzwerk

<sup>12</sup> Damit ein Programm ausführbar ist, muss es diese Ausführrechte erhalten.

<sup>13</sup> Der Pfad (hier `./` zum aktuellen Verzeichnis) muss immer angegeben werden.

<sup>14</sup> Bestimmte Befehle benötigen Administratorrechte. Dann muss vor den Befehl das Prefix `sudo` gestellt werden, worauf – zumindest bei der ersten Verwendung (während einer Session) von `sudo` – das Passwort `hslu` (Kleinbuchstaben) abgefragt wird.

<sup>15</sup> In Kapitel 7.1 sind die Schritte zum Einstellen der Netzwerk Settings der VirtualBox beschrieben.

Interfaces angezeigt werden, sowie das Local Loopback Interface auf der Adresse 127.0.0.1/8.

```

oscar@oscar-VirtualBox: ~
File Edit View Search Terminal Help
oscar@oscar-VirtualBox:~$ ifconfig
eth0: Link encap:Ethernet HWaddr 08:00:27:e1:9c:e1
      inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
      inet6 addr: fe80::a00:27ff:fe91:9ce1/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:257 errors:0 dropped:0 overruns:0 frame:0
      TX packets:1008 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:41579 (41.5 KB) TX bytes:110524 (110.5 KB)

eth1: Link encap:Ethernet HWaddr 08:00:27:91:a6:84
      inet addr:192.168.56.101 Bcast:192.168.56.255 Mask:255.255.255.0
      inet6 addr: fe80::a00:27ff:fe91:a684/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:419 errors:0 dropped:0 overruns:0 frame:0
      TX packets:379 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:46290 (46.2 KB) TX bytes:63159 (63.1 KB)

lo: Link encap:Local Loopback
     inet addr:127.0.0.1 Mask:255.0.0.0
     inet6 addr: ::1/128 Scope:Host
     UP LOOPBACK RUNNING MTU:65536 Metric:1
     RX packets:462 errors:0 dropped:0 overruns:0 frame:0
     TX packets:462 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:0
     RX bytes:36329 (36.3 KB) TX bytes:36329 (36.3 KB)

oscar@oscar-VirtualBox:~$

```

**Abbildung 16:** Netzwerk Interface der Raspberry Pi-SDK VM.

Weiterhin kann mittels `netstat -r` die Routing Tabelle angezeigt werden und sollte wie in Abbildung 17 gezeigt aussehen. Die Default Route sollte auf das NAT Interface (IP 10.0.2.2/24 auf `eth0`) zeigen. Falls dies nicht der Fall ist hilft in der Regel ein Neustart der Netzwerk Interfaces mittels

`sudo service network-manager restart.`

Im „worst case“ über das VirtualBox Menü einen Reboot ausführen.

```

oscar@oscar-VirtualBox: ~
File Edit View Search Terminal Help
oscar@oscar-VirtualBox:~$ netstat -r
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
default 10.0.2.2 0.0.0.0 UG 0 0 0 eth0
10.0.2.0 * 255.255.255.0 U 0 0 0 eth0
link-local * 255.255.0.0 U 0 0 0 eth0
192.168.56.0 * 255.255.255.0 U 0 0 0 eth1
oscar@oscar-VirtualBox:~$

```

**Abbildung 17:** Routing Tabelle der Raspberry Pi-SDK VM.

Abschliessend kann die Funktionalität der Netzwerk Interfaces aus der Raspberry Pi-SDK VM heraus noch getestet werden:

1. Öffnen des Mozilla Browsers zum Test für den Internet Zugriff.
2. `ssh pi@192.168.1.10:`  
Ebenso wie der Host (s. Abschnitt 2.3.3) kann auch die Raspberry Pi-SDK VM eine SSH-Verbindung zum Raspberry Pi herstellen. Im Gegensatz zu Windows



ist allerdings der SSH-Client als Standardprogramm in der Ubuntu Distribution enthalten und kann mittels `ssh` von der Kommandozeile aus gestartet werden. Nach der Eingabe des Passwortes<sup>16</sup> `hslu` (Kleinbuchstaben) wird ein Terminal zur Remoteverbindung zum Raspberry Pi geöffnet.

3. `ping 192.168.1.11:`  
Senden eines Ping Request an den Host auf die IP 192.168.1.11, welche mit den unter Abschnitt 2.3.1 vorgenommenen Einstellungen übereinstimmen muss.

### 3.2.2. Datenaustausch via „Gemeinsamer Ordner“

Der „Gemeinsame Ordner“ wurde im Host Betriebssystem bei der Konfiguration der Raspberry Pi-SDK VM definiert, womit dessen Position klar ist (Abbildung 14). In der Raspberry Pi-SDK VM ist der Ordner im Pfad `/media/xxx` zu finden, wobei `xxx` dem Namen des Verzeichnisses im Host entspricht (ebenfalls im Konfigurationsdialog (Abbildung 14) angezeigt). Zum Test öffnen Sie in der VM einen File Explorer (Abbildung 18, blauer Kreis) und zeigen Sie via `Computer` den Inhalt des `/media/xxx` an. Sie sollten die gleichen Files sehen, wie auch vom Host aus. Als abschliessenden Test können Sie noch ein gemeinsames File ändern (im Host oder in der Raspberry Pi-SDK VM) und die Änderung auf der jeweils „anderen Seite“ verifizieren.

Sollten die Files im Host Betriebssystem von der VM aus nicht sichtbar sein, dann hilft in der Regel der Befehl:

```
sudo mount -t vboxsf xxx /media/xxx
```

(c.f. Abschnitt 9.4.3)

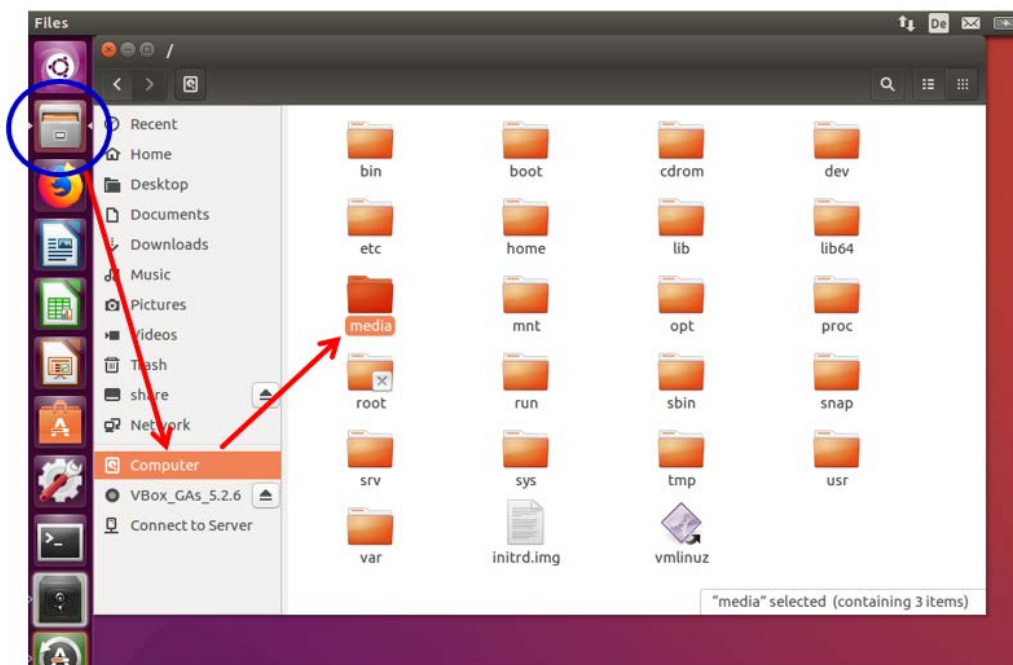
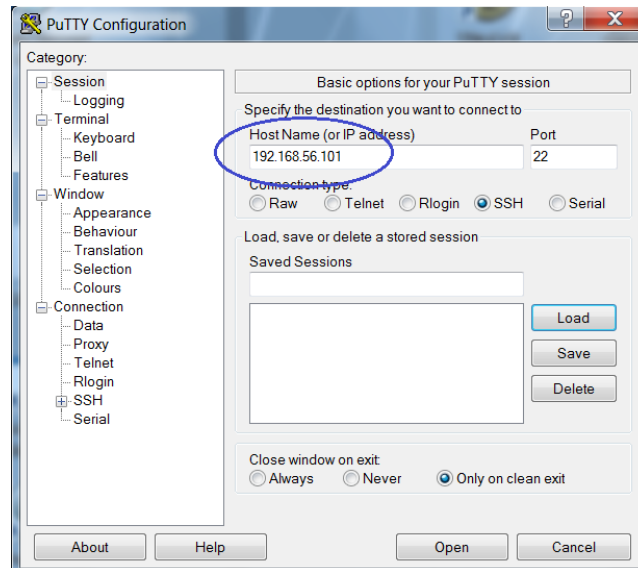


Abbildung 18: Aufsuchen des „Gemeinsamen Ordners“ in der VM.

<sup>16</sup> Der Benutzername `pi` ist bereits durch den Aufruf `pi@192.168.1.10` definiert.

### 3.2.3. Kommunikation von Host zu VM via SSH

Eine SSH Verbindung vom Host zur Raspberry Pi-SDK VM kann interessant sein, um z.B. das lästige Switchen von Host zu VM zu vermeiden oder um Zugriff auf Repositories (svn oder git) auf der Raspberry Pi VM vom Host aus zu haben. Zum Test der SSH-Verbindung wird analog zu den Schritten in Abschnitt 2.3.3 das Programm Putty geöffnet und eine Verbindung zur IP-Adresse der Raspberry Pi-SDK VM via der IP 192.168.56.101 hergestellt. Dabei sind der Benutzername `pi` und das Passwort `hslu` (beides Kleinbuchstaben) einzugeben.



**Abbildung 19:** Herstellen einer SSH-Verbindung vom Host zur Raspberry Pi-SDK VM.

## 4. Verwendung des Raspberry Pi SDK

Die grundsätzliche Verwendung der Raspberry Pi Plattform ist in zahlreichen Tutorials beschrieben, ebenso gibt es zahlreiche Beispielprogramme für die Erstellung von Bildverarbeitungsanwendungen auf dem Raspberry Pi [6]. Hier stellen wir für einen einfachen Einstieg ein C++ SDK zu Verfügung, welches die wesentlichen Funktionalitäten wie Bildakquisition und Weitergabe an den Host übernimmt. Für die eigentlichen Bildverarbeitungsroutinen verwenden wir die Open Source Bibliothek OpenCV, welche sich als Quasistandard für Bildverarbeitung im universitären Bereich etabliert hat [7]. Um die grundsätzlichen Zusammenhänge der verschiedenen Komponenten klar zu illustrieren, wollen wir im Folgenden einige Beispielanwendungen entwickeln, die in aufsteigender Komplexität die Verwendung des SDK erläutert. Sie können, je nach Vorwissen, die ersten Punkte überspringen und weiter unten „einsteigen“.

### 4.1. Ein erstes Hello World Programm

Wir werden das „Hello World“ Programm rein mit Hilfe der Kommandozeile entwickeln, vor allem auch, um etwas Übung mit der Linux Kommandozeile zu gewinnen. Es ist grundsätzlich aber möglich, die folgenden Code Beispiele per Copy-Paste in den Kommandozeilen Editor zu übernehmen.

#### 4.1.1. Übung 1: Entwicklung eines „Hello World“ Programms für Linux in C

##### Ziele:

1. Wir haben eine erste Übung im Umgang mit der Linux Kommandozeile
2. Wir können ein einfaches File editieren und ändern
3. Wir können ein C-File für Linux kompilieren und linken, um einen ausführbaren Code zu erzeugen.
4. Wir können den Binärcode ausführen.

##### Übung im Detail:

1. Wechseln Sie auf der Raspberry Pi-SDK VM in das home-Verzeichnis des Users `hslu`<sup>17</sup>:

```
cd ~
```

Verifizieren Sie mittels

```
pwd
```

ob es funktioniert hat. Wechseln Sie nun in das Verzeichnis `uebung`:

```
cd uebung
```

---

<sup>17</sup> Die Tilde `~` ist ein Shortcut für das Home-Verzeichnis.

2. Erstellen Sie ein neues Verzeichnis `uebung01` unter `/home/pi/uebung`:

```
mkdir uebung01
```

Wechseln Sie in das Verzeichnis.

3. Erzeugen Sie mit Hilfe eines Editors (`gedit`) ein kleines C-Source File mit Namen `hello_world.c`:

```
gedit hello_world.c
```

Es öffnet sich ein Fenster, in dem Sie „normal“ editieren können. Im Folgenden ist ein kleines Beispielprogramm für einen einfachen Task. Sie können es direkt übernehmen (Copy-Paste), oder aber selbst eine Anwendung entwickeln:

```
#include <stdio.h>

int main(const int argc, const char * argv[])
{
    int i;

    printf("hello C-world\n");
    for(i = 0; i < argc; i++) {
        printf("the %d-th argument is %s\n", i, argv[i]);
    }
    return(0);
}
```

4. Erstellen Sie aus diesem C-Quellcode einen ausführbaren Code für das Linux Betriebssystem mittels

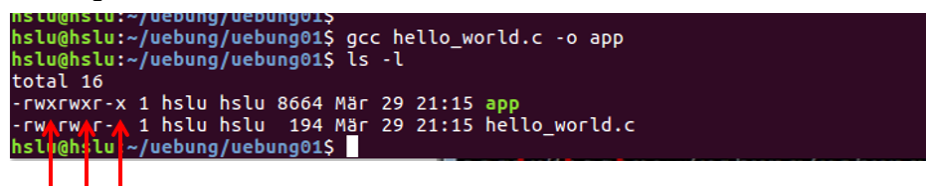
```
gcc hello_world.c -o app
```

Die Option `-o` definiert, dass der ausführbare Code `app` genannt wird.

5. Verifizieren Sie auf der Kommandozeile die Tatsache, dass die Applikation `app` ausführbar ist:

```
ls -l
```

Das Ergebnis sollte so aussehen:



```
hslu@hslu:~/uebung/uebung01$ gcc hello_world.c -o app
hslu@hslu:~/uebung/uebung01$ ls -l
total 16
-rwxrwxr-x 1 hslu hslu 8664 Mär 29 21:15 app
-rw-rw-r-- 1 hslu hslu 194 Mär 29 21:15 hello_world.c
hslu@hslu:~/uebung/uebung01$
```

Das Flag `x` (für ausführbar<sup>18</sup>) ist für alle drei Benutzerklassen<sup>19</sup> (user/group/others) gesetzt<sup>20</sup>.

<sup>18</sup> 'r' bzw. 'w' stehen für Lese- (read) und Schreib- (write) Rechte.

<sup>19</sup> Für Details siehe hierzu <http://de.wikipedia.org/wiki/Unix-Dateirechte#Benutzerklassen>.

<sup>20</sup> Um das Flag `x` manuell zu setzen, verwenden Sie `chmod a+x Filename`.

6. Führen Sie das Programm aus mittels:

```
./app
```

Testen Sie (falls Sie obiges Beispiel verwendet haben) dessen Funktionalität, z.B.:

```
./app 1.Argument 2.Argument noch_ein_Argument
```

### 4.1.2. Übung 2: Entwicklung eines „Hello World“ Programms für Raspbian in C

#### Ziele:

1. Wir können ein C-File kompilieren und linken, um einen ausführbaren Code für das Raspbian Betriebssystem des Raspberry Pi zu erzeugen.
2. Wir können den Binärcode für Raspbian auf die Raspberry Pi übertragen und ausführen.

#### Übung im Detail:

1. Erstellen Sie ein neues Verzeichnis `uebung02` unter `/home/pi/uebung` und wechseln Sie in das Verzeichnis.
2. Kopieren Sie das File `hello-world.c` (aus Abschnitt 4.1.1) in dieses Verzeichnis<sup>21</sup>:

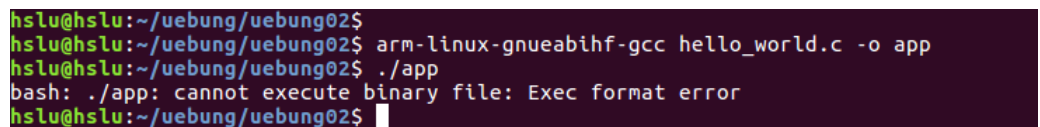
```
cp ../uebung01/hello_world.c .
```

Verifizieren Sie das Ergebnis.

3. Erstellen Sie aus `hello-world.c` einen ausführbaren Code für Raspbian mit Hilfe des Cross Compilers:

```
arm-linux-gnueabihf-gcc hello_world.c -o app
```

4. Verifizieren Sie wieder die Zugriffsrechte auf des Binärfiles (`app`). Wenn Sie nun versuchen, die Datei `app` (auf der Ubuntu VM) auszuführen, dann erhalten Sie eine Fehlermeldung:



```
hslu@hslu:~/uebung/uebung02$  
hslu@hslu:~/uebung/uebung02$ arm-linux-gnueabihf-gcc hello_world.c -o app  
hslu@hslu:~/uebung/uebung02$ ./app  
bash: ./app: cannot execute binary file: Exec format error  
hslu@hslu:~/uebung/uebung02$
```

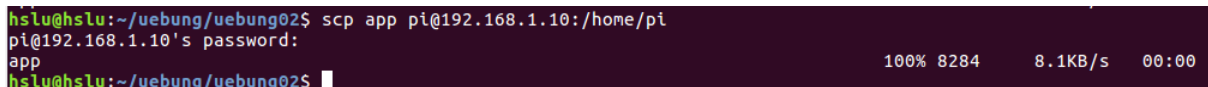
Denn das Binärfile ist nun für die Raspberry Pi Architektur kompiliert und nur dort lauffähig.

<sup>21</sup> Ein Punkt im Filepfad (also `.` oder `./`) bezeichnet das aktuelle Verzeichnis, zwei Punkte im Filepfad (also z.B. `../`) bezeichnen das über dem aktuellen Verzeichnis liegende Verzeichnis.

5. Übertragen Sie daher das Binärfile (`app`) auf den Raspberry Pi. Dies erfolgt mittels der Applikation `scp`:<sup>22</sup>

```
scp app pi@192.168.1.10:/home/pi
```

Zur Syntax: `scp` überträgt das File `app` (im lokalen Verzeichnis<sup>23</sup>) auf den Rechner mit der IP-Adresse `192.168.1.10`. Dabei verwendet `scp` für die Anmeldung auf dem Remote Computer `192.168.1.10` den Benutzer `pi` und kopiert die Datei dort in das Verzeichnis `/home/pi`. Nach dem Ausführen des Befehls, werden Sie nach dem Passwort des Benutzers `pi` auf dem Remote Computer `192.168.1.10` gefragt. Dieses ist – wie für die SSH-Verbindung auf zum Raspberry Pi (Kapitel 2.3.3) schon verwendet – `hslu` (Kleinbuchstaben). Es sollte eine Status Meldung wie folgt angezeigt werden:



```
hslu@hslu:~/uebung/uebung02$ scp app pi@192.168.1.10:/home/pi
pi@192.168.1.10's password:
app
hslu@hslu:~/uebung/uebung02$
```

100% 8284 8.1KB/s 00:00

6. Verbinden Sie sich nun per SSH zum Raspberry Pi:

```
ssh pi@192.168.1.10
```

Ebenso können Sie mit Putty gem. Kapitel 2.3.3 vom Host aus arbeiten. Verifizieren Sie, dass die Datei `app` im Verzeichnis `/home/pi` vorhanden ist und starten Sie das Programm.

7. Löschen Sie die Datei `app` auf dem Raspberry Pi mittels:

```
rm app
```

Dann schliessen Sie am Ende die SSH-Verbindung wieder.

### 4.1.3. Übung 3: Entwicklung eines „Hello World“ in C++

#### Ziele:

- Wir kennen den Unterschied zwischen `gcc` und `g++`.

#### Übung im Detail:

- Erstellen Sie ein neues Verzeichnis `uebung03` unter `/home/pi/uebung` und wechseln Sie in das Verzeichnis.
- Erzeugen Sie mit Hilfe eines Editors (`gedit`) ein kurzes C++-Quell File `hello_world.cpp`. Im Folgenden wieder ein Beispiel, welches Sie direkt übernehmen können, aber Sie können auch selbst eine Anwendung entwickeln:

```
#include <stdio.h>
```

<sup>22</sup> Bedenken Sie, dass Sie den Raspberry Pi nur über das IP-Netzwerk erreichen können und z.B. kein gemeinsames Verzeichnis zum Dateiaustausch haben.

<sup>23</sup> `scp` kann auch zum Filetransfer zwischen zwei verschiedenen Remote Computern verwendet werden.

```
class test
{
public:
    test(int A, int B): a(A), b(B) {}
    ~test(){}

    int a;
    int b;
};

int main(const int argc, const char * argv[])
{
    test t(1,2);

    printf("hello C++ world\n");
    printf("class members are: a = %d, b = %d\n", t.a, t.b);

    return(0);
}
```

3. Erstellen Sie einen ausführbaren Code für dieses C++-Programm nun sowohl für das Linux Betriebssystem als auch für das Raspbian Betriebssystem mittels

```
g++ hello_world.cpp -o app
```

bzw.

```
arm-linux-gnueabihf-g++ hello_world.cpp -o app
```

Testen Sie auch, dass die Verwendung des C-Compilers `gcc` in einem Fehler resultiert.

4. Verifizieren Sie wieder die Funktionalität des Programmes, sowohl auf der Raspberry Pi-SDK VM als auch auf dem Raspberry Pi selbst (analog 4.1.1 bzw. 4.1.2).

#### 4.1.4. Übung 4: Erstellung und Einbindung einer Library/Skripting

##### Ziele:

1. Wir kennen den Unterschied zwischen dem Kompilieren und dem Linken und dem Bezug zu `#include` Statements sowie dem Einbinden von externen Libraries.
2. Wir können Libraries erstellen und in ein Programm einbinden.
3. Wir können ein einfaches Shell-Skript erstellen.

##### Übung im Detail:



1. Erstellen Sie ein neues Verzeichnis `uebung04` unter `/home/pi/uebung` und wechseln Sie in das Verzeichnis.
2. Erzeugen Sie mit Hilfe eines Editors (`gedit`) die folgenden drei C-Source Files (\*.c, A – C) und das Header File (\*.h, D):

A) `func_add.c`:

```
double add_d(double a, double b)
{
    return (a+b);
}
```

B) `func_mul.c`:

```
double mul_d(double a, double b)
{
    return (a*b);
}
```

C) `hello_math.c`:

```
#include <stdio.h>
#include <stdlib.h>
#include "func.h"

int main(const int argc, const char * argv[])
{
    double i1 = 1.1;
    double i2 = 3.4;

    if(argc > 1)
    {
        i2 = atof(argv[1]);
    }

    printf("hello C-world\n");
    printf("sum of %2.3f, %2.3f is %2.3f\n", i1, i2, add_d(i1, i2));
    printf("product of %2.3f, %2.3f is %2.3f \n", i1, i2, mul_d(i1, i2));
    return(0);
}
```

D) `func.h`:

```
double add_d(double a, double b);

double mul_d(double a, double b);
```

3. Wir wollen nun aus den beiden Source Files `func_add.c` und `func_mul.c` eine Library erstellen, welche von unserem main-File `hello_math.c` verwendet wird<sup>24</sup>. Studieren Sie den Code von `hello_math.c` und beachten

---

<sup>24</sup>Libraries werden verwendet, um komplexe Funktionalität zur Verfügung zu stellen, ohne den Source Code publizieren zu müssen. Ausserdem bieten Sie die Möglichkeit, umfangreiche Projekte zu strukturieren. Sie bestehen häufig aus hunderten von Source Files.

Sie vor allem das

```
#include "func.h"
```

Statement. Wieso braucht es dieses?

4. Wir werden nun die Erstellung des Binärcodes von `hello_math.c` in zwei Schritte unterteilen:

- a. Wir kompilieren das Source File und erstellen ein sog. Objektfile. Dieses stellt noch keinen ausführbaren Code dar:

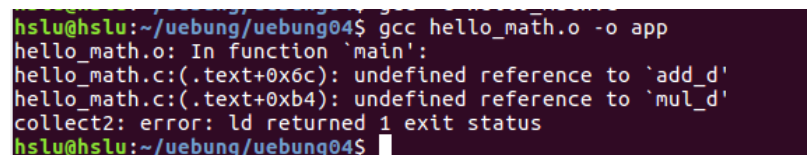
```
gcc -c hello_math.c
```

Das Flag `-c` weist den Compiler an, den Source Code nur zu kompilieren. Sie werden feststellen, dass das Resultat ein sog. Objektfile (`hello_math.o`) ist.

- b. Nun linken wir das Objektfile mit den notwendigen Libraries<sup>25</sup>, um einen ausführbaren Code zu erstellen. Bisher hatten wir diese beiden Schritte nicht unterschieden und `gcc` (bzw. `g++`) führt per Default beide „in einem Aufwasch“ durch. Zum Linken geben Sie einfach das Objektfile (allgemein eine Liste von Objektfiles) an:

```
gcc hello_math.o -o app
```

Allerdings werden Sie feststellen, dass die obige Anweisung in einem Fehler resultiert:



```
hslu@hslu:~/uebung/uebung04$ gcc hello_math.o -o app
hello_math.o: In function 'main':
hello_math.c:(.text+0x6c): undefined reference to 'add_d'
hello_math.c:(.text+0xb4): undefined reference to 'mul_d'
collect2: error: ld returned 1 exit status
hslu@hslu:~/uebung/uebung04$
```

Können Sie sich denken, woher der Fehler stammt?

Das `#include "func.h"` Statement weist den Compiler auf die Definition der Funktionen `add_d()` und `mul_d()` hin, d.h. im Kompilierungsschritt kann der Compiler die korrekte Verwendung der Funktionen bez. Syntax und Typ überprüfen. Aber die eigentliche Implementierung, liegt dem Compiler noch nicht vor<sup>26</sup>. Diese braucht er auch erst beim Linken, nämlich dann, wenn er den ausführbaren Code erzeugen will.

Daher müssen wir beim Linken noch die Objektfiles mit der Implementierung der `add_d()` und `mul_d()` Funktionen angeben.

Häufig sind solche externen Objektfiles in Bibliotheken (Libraries) zusammengefasst, die mit der Dateiendung `*.a` gekennzeichnet sind. Hier wollen wir nun unsere eigene Library erzeugen, welche wir

---

Die Verwendung/Bildung einer Library hat hier rein didaktische Zwecke. Für das gezeigte Beispiel könnten die drei Source Files natürlich einfach direkt zusammen zur fertigen Applikation kompiliert und gelinkt werden.

<sup>25</sup> So greift der Linker z.B. auf die C-Standard Libraries zu.

<sup>26</sup> An dieser Stelle können Sie auch verifizieren, was passiert, falls Sie das `#include "func.h"` statement auskommentieren. Hinweis: Der Compiler nimmt per Default an, dass der Input und Rückgabe Wert einer Funktion vom Typ `int` ist.

`libfunc.a` nennen.

5. Zum Erstellen der Library müssen wir zuerst die beiden Source Files `func_add.c` und `func_mul.c` kompilieren, um die entsprechenden Objektfiles zu erstellen. Hierzu verfahren wir wie für das File `hello_math.c`:  

```
gcc -c func_add.c
gcc -c func_mul.c
```

Zur Erstellung der Library aus den Objektfiles `func_add.o` und `func_mul.o` dient das Programm `ar`. Führen Sie hierzu folgenden Befehl aus:

```
ar rc libfunc.a func_add.o func_mul.o
```

Dieser weist das Programm `ar` an, aus den Objektfiles `func_add.o` und `func_mul.o` die Library `libfunc.a` zu erstellen. Die Rolle der Argumente `rc` können Sie mit Hilfe von

```
ar --help
```

nachvollziehen.

Verifizieren Sie, dass die Library `libfunc.a` erstellt wurde.

6. Nun können wir den Befehl zum Linken (s. Punkt 4) ergänzen. Dort fehlte noch die Implementierung der Funktionen `add_d()` und `mul_d()`. Diese können wir mittels der Library `libfunc.a` hinzufügen:

```
gcc hello_math.o libfunc.a -o app
```

Nun erfolgt das Linken ohne Fehler.

7. Verifizieren Sie die Funktionalität des Programms.
8. Schliesslich wollen wir uns die Arbeit noch etwas abkürzen und das vielleicht mächtigste Tool des Linux Betriebssystems kennenlernen, die Shell-Skripte. Damit können Sie praktisch jeden Vorgang automatisieren, was die immense Stärke dieses Tools ausmacht. Als einfache Einführung sei folgendes Shell Skript gedacht, mit welchem wir die eben durchgeführten Schritte nun mit Hilfe des Cross Compilers für das Target durchführen:

```
#!/bin/bash
```

```
echo "compile func_add and func_mul"
arm-linux-gnueabihf-gcc -c func_add.c
arm-linux-gnueabihf-gcc -c func_mul.c
```

```
echo "create library file libfunc.a from object "
arm-linux-gnueabihf-ar rc libfunc.a func_add.o func_mul.o
```

```
echo "compile $1.c to give object file $1.o"
arm-linux-gnueabihf-gcc -c $1.c
```

```
echo "link $1.o and libfunc.a to file app"
arm-linux-gnueabihf-gcc $1.o libfunc.a -o app
```

Für dessen Verständnis müssen Sie nur Folgendes wissen:

- Das `#!/bin/bash` Statement legt fest, welcher Interpreter verwendet werden soll. In einem `perl` Skript würde hier z.B. `perl` stehen.
- Ein Shell-Skript läuft grundsätzlich von oben nach unten ab (dies ist z.B. bei Makefiles nicht der Fall).
- Argumente, welche beim Aufruf an das Skript übergeben werden, können mit `$1`, `$2`, usw. verwendet werden. Dabei gibt der Index die Position des Arguments in der List an.
- Mit `echo` kann Information auf die Kommandozeile ausgegeben werden.
- Im Prinzip können alle Befehle der Linux Kommandozeile verwendet werden.

Studieren Sie das Skript und erstellen Sie es in einem File `cl.sh`. Die Endung `sh` ist die Standard Fileextension für Shell Skripte. Machen Sie das Skript ausführbar, mittels:

```
chmod a+x cl.sh
```

Testen Sie das Skript mittels:

```
./cl.sh hello_math
```

Beachten Sie dabei, dass das Inputfile OHNE Extension (`.c`) angegeben wurde, da diese im Skript ergänzt wird.

Verifizieren Sie die Funktionalität der Applikation, indem Sie diese mittels `scp` auf das Target übertragen und ausführen.

Nachdem wir nun einen ersten Einstieg in die Programmierung von C(++) unter Linux für die Raspberry Pi geschafft haben, wenden wir uns nun unserem eigentlichen Interesse an der Raspberry Pi zu, nämlich der Bildverarbeitung. Wie bereits einführend erwähnt, wollen wir hierfür die OpenCV Bibliothek verwenden und wollen uns daher im Folgenden etwas damit vertraut machen.

## 4.2. Anwendungsbeispiele für OpenCV Library

Die einfachste Methode, sich mit dem Framework vertraut zu machen, ist einige Anwendungsbeispiele zu studieren. Wir werden auch hier wieder mit steigender Komplexität vorgehen.

### 4.2.1. Übung 5: Einbindung der OpenCV Library

#### Ziele:

1. Wir verstehen erste grundlegende SW-Komponenten einer OpenCV-basierten Anwendung.
2. Wir können die OpenCV Library in ein eigenes Programm einbinden.
3. Wir können Daten vom Raspberry Pi auf den Host übertragen.

Übung im Detail:

1. Erstellen Sie ein neues Verzeichnis `uebung05` unter `/home/pi/uebung` und wechseln Sie in das Verzeichnis.
2. Unter `/home/pi/uebung` liegt das File `uebung05.tar`. Legen Sie dies im o.g. Verzeichnis ab. Entpacken Sie das tar-Archiv mit:

```
tar -xf uebung05.tar
```

Machen Sie sich in diesem Zusammenhang auch mit<sup>27</sup>

```
man tar
```

(unter `Examples`) mit der grundsätzlichen Verwendung von `tar` vertraut.  
*Hinweis:* Die Navigation auf den Man-Pages erfolgt z.B. mit `↑` und `↓`, zum Verlassen einfach die `q`-Taste drücken.

3. Studieren Sie das C-Quellcode File `inv_img.cpp`. Vollziehen Sie – ggf. mit Hilfe der OpenCV-Dokumentation [8] – die grundsätzliche Funktionsweise des Programmes nach.
4. Studieren Sie das Shell Skript `cl.sh`, welches für die Erstellung des Binärcodes verwendet werden kann. Dort werden zuerst verschiedene Pfad Variablen definiert, z.B.

```
INC=../../image_proc_agent/..
```

welche dann im Skript gemäss `${INC}` referenziert werden können. Machen Sie sich insbesondere mit folgenden Compiler- bzw. Linker Optionen vertraut:

```
-I${INC}:
```

Definition eines include Verzeichnisses, hier für die OpenCV Library.

```
-O2: Optimierung des Code auf Geschwindigkeit
```

Analysieren Sie weiterhin, wie die verschiedenen OpenCV und 3rd Party Libraries über die Pfadangaben `${LIB}` bzw. `${LIBX}` beim Linken eingebunden werden.

5. Erstellen Sie nun die Applikation für den Host mittels

```
./cl.sh inv_img
```

und führen Sie sie aus. Verifizieren Sie mit dem File Explorer ob das Ausgabefile `img_inv.png` dem erwarteten Ergebnis entspricht.

6. Nun werden wir die Applikation für das Raspberry Pi Target erstelle mit dem Skript `cl_arm.sh`. Dieses Skript ähnelt dem bereits verwendeten Skript

---

<sup>27</sup> Grundsätzlich kann unter Linux mit dem Befehl `man Befehlsname` (Manual Page) oder mittels `Befehlsname --help` (2 Minuszeichen!) Hilfe zu einem Befehl erhalten werden,

`cl.sh`, wobei die Linker Optionen etwas verkürzt wurden. Einerseits wurden mit der Linker Option `-L` Suchpfade für die Libraries definiert. Zusätzlich wurden die Libraries nun mit der Linker Option `-l` angegeben. Dabei ist die Option, dass der Linker beim Namen hinter der `-l` Option das `lib` voranstellt und die Endung `.a` anhängt.

Erstellen Sie die Applikation mit

```
./cl_arm.sh inv_img
```

und übertragen Sie sie (inklusive Bild) auf den Raspberry Pi:

```
scp London.png app pi@192.168.1.10:/home/pi/tmp
```

Verbinden Sie sich mit SSH auf den Raspberry Pi und führen die Applikation aus. Verifizieren Sie, ob danach im Verzeichnis `/home/pi` das Ausgabebild `img_inv.png` enthalten ist.

- Übertragen Sie nun das Ausgabebild vom Raspberry Pi auf den Host. Dies machen Sie vom **Host** aus! Erinnern Sie sich an die Verwendung von `scp`<sup>28</sup>:

```
scp pi@192.168.1.10:/home/pi/tmp/img_inv.png .
```

D.h. auch die Quelle kann ein Remote Computer sind.

Damit haben wir unsere erste wirkliche Applikation im Zusammenhang mit Bildverarbeitung auf dem Raspberry Pi erstellt.

Sie könnten nach dieser Stelle schon einmal versuchen, die Bilddaten vor dem Abspeichern in anderer Form zu manipulieren.

#### 4.2.2. Übung 6: Verwendung des Boa Webservers

Mit der vorigen Übung (4.2.1) haben wir unser Ziel einer Bildverarbeitung auf dem Raspberry Pi Plattform eigentlich schon fast erreicht. Allerdings ist die Visualisierung der Bilddaten mittels `scp` noch recht mühsam und sehr langsam. Hier wäre ein live Zugriff wünschenswert, wie er z.B. bei IP-Kameras möglich ist. In dieser Übung wollen wir die Verwendung eines Webservers zur Bilddatenübertragung vom Raspberry Pi verstehen.

##### Ziele:

- Wir verstehen die grundlegende Arbeitsweise des Boa Webservers.
- Wir können Bilddaten über den Webserver herunterladen.

##### Übung im Detail:

- Erstellen Sie ein neues Verzeichnis `uebung06` unter `/home/pi/uebung` und wechseln Sie in das Verzeichnis.
- Unter `/home/pi/uebung` liegt das File `uebung06.tar`. Legen Sie dies im o.g. Verzeichnis ab. Entpacken Sie das tar-Archiv.  
Das Archiv enthält wieder ein C++ Source File (`grab_img.cpp`), ein Shell

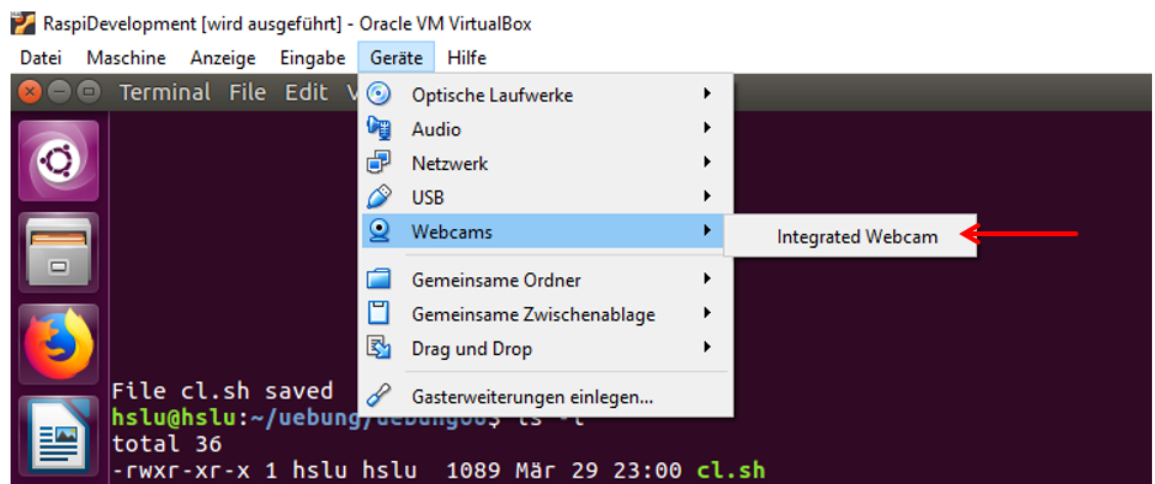
<sup>28</sup> Übersehen Sie nicht den Punkt am Ende des Befehls für die Angabe des Zielverzeichnisses.

Skript `cl.sh` zum Kompilieren sowie zwei Text- (`file1.txt`, `file2.txt`) und ein html-File `index.html`.

- Studieren Sie wieder das Source File und versuchen Sie nachzuvollziehen, was bei dessen Ausführung passiert.
- Studieren Sie auch die Änderungen im Skript `cl.sh`: Der arm-Compiler kann nun über das 2. Kommandozeilen Argument `$2` gewählt werden und der Filetransfer wird nach dem Erstellen des Binärcodes auch über das Skript ausgeführt. Zusätzlich zur Applikation werden auch das html-File `my_index.html` und die zwei Textfiles übertragen.
- Zuerst erstellen wir die Applikation für den Host mittels:

```
./cl.sh grab_img
```

- Um die Applikation erfolgreich auf dem Host ausführen zu können, benötigen wir eine Kamera. Wenn Sie eine (integrierte) Webcam haben, dann können Sie diese zur VM hinzufügen<sup>29</sup>:



- Führen Sie nun die Applikation aus und verifizieren Sie, dass die Ausgabebilder in das aktuelle Verzeichnis geschrieben werden (`out00.bmp`, `out001.bmp`, ...).
- Nun erstellen wir die Applikation für das Raspberry Pi Target und übertragen die Files direkt mittels:

```
./cl.sh grab_img arm
```

Dabei ist für die scp-Verbindung das Passwort einzugeben.

- Werfen Sie einen Blick auf das html-File. Es ist denkbar „schlank“ gehalten und besteht im Wesentlichen nur aus zwei Links zu Textfiles

```
<a href="file1.txt">file1.txt</a><br>
```

<sup>29</sup> Falls nicht, dann fahren Sie beim übernächsten Punkt fort.



und einer Liste von Bildern:

```

<br>
...
```

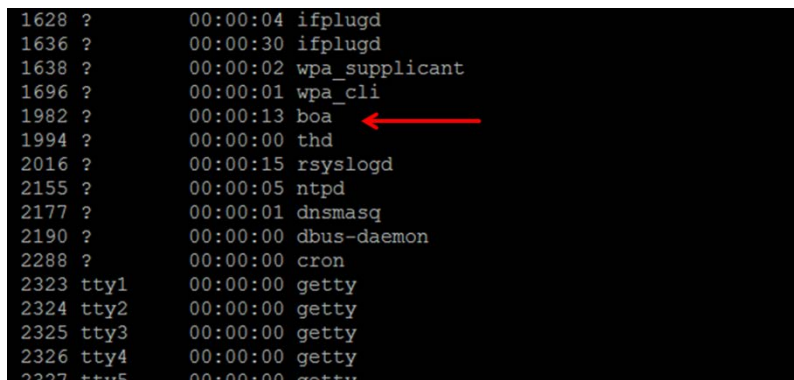
Wie Ihnen sicher auffällt, entsprechen die Bildnamen gerade den von der Applikation abgespeicherten Bildern.

Zusätzlich ist noch der Meta Tag im html-Header interessant:

```
<meta http-equiv="refresh" content="10">
```

Dieser bewirkt, dass die Seite alle 10 Sekunden neu geladen wird, d.h. die Bilder, falls vorhanden, ersetzt werden.

10. Verbinden Sie sich nun mit SSH auf den Raspberry Pi und geben Sie das Kommando `ps -A` ein. Es zeigt alle laufenden Prozesse an. Unter anderem finden Sie in der Liste den Prozess `boa`, einen schlanken Webserver speziell auch für embedded Anwendungen [4].



```
1628 ?      00:00:04 ifplugd
1636 ?      00:00:30 ifplugd
1638 ?      00:00:02 wpa_supplicant
1696 ?      00:00:01 wpa_cli
1982 ?      00:00:13 boa ←
1994 ?      00:00:00 thd
2016 ?      00:00:15 rsyslogd
2155 ?      00:00:05 ntpd
2177 ?      00:00:01 dnsmasq
2190 ?      00:00:00 dbus-daemon
2288 ?      00:00:00 cron
2323 tty1    00:00:00 getty
2324 tty2    00:00:00 getty
2325 tty3    00:00:00 getty
2326 tty4    00:00:00 getty
2327 tty5    00:00:00 getty
```

11. Lassen Sie sich den Inhalt der Konfigurationsdatei von `boa` anzeigen, mittels:

```
less /etc/boa/boa.conf
```

Dabei können Sie mit den Pfeiltasten  $\uparrow\downarrow$  navigieren (q-Taste zum verlassen). Verschiedene Einträge sind interessant:

- Port 80:  
Wie jeder „anständige“ Webserver „horcht“ auch `boa` auf Port 80.
- ErrorLog `/var/log/boa/error_log`  
Das Error Log File, welches bei Problemen mit der Verbindung zwischen Browser und Webserver immer ein erster Anhaltspunkt ist.
- DocumentRoot `/var/www`  
Das Root Verzeichnis der html Dokumente, die vom Webserver veröffentlicht werden.

- d. `DirectoryIndex index.html`  
Index der verfügbaren Verzeichnisse oder Files.

## 12. Wechseln Sie in das Verzeichnis

```
/var/www/
```

und verifizieren Sie die Präsenz des html-Files `index.html`. In unserem – sehr einfachen Fall – enthält das File `index.html` nur den Link auf das html File `VcaWebGui.html`, welches für die Anzeige der Bilder im Browser (Abbildung 7) verantwortlich ist. Dieses ist allerdings schon relativ komplex, weshalb wir mit dem einfachen Beispiel `my_index.html` arbeiten wollen.

## 13. Wechseln Sie in das Verzeichnis

```
/home/pi/tmp
```

in welches wir die Applikation und die restlichen Files kopiert hatten. Kopieren Sie nun das html- und die Text-File in das Verzeichnis `/var/www` mittels:

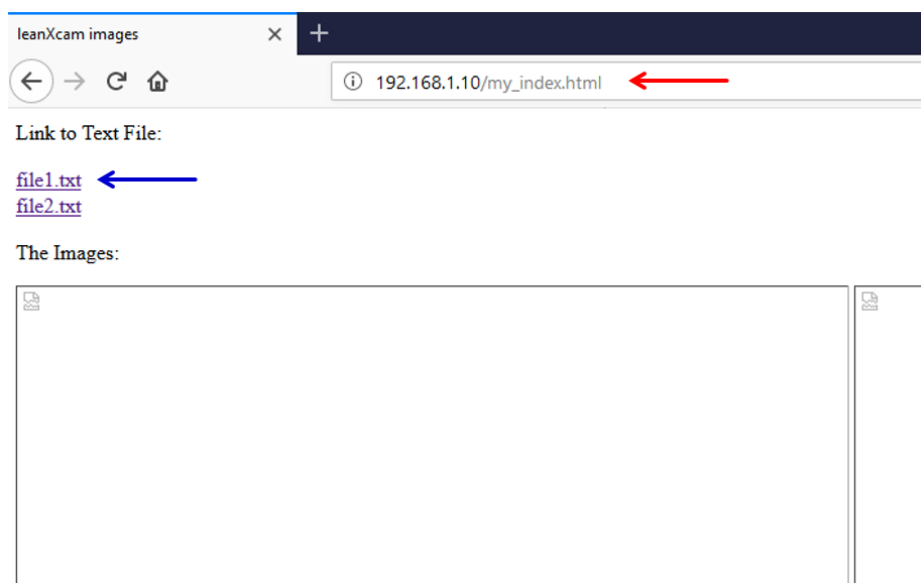
```
sudo cp file* my_index.html /var/www
```

Dabei benötigen Sie root-Rechte, das der User pi keinen Schreibzugriff auf dieses Verzeichnis hat.

## 14. Öffnen Sie nun einen Browser und verbinden Sie sich auf den Raspberry Pi (analog Abbildung 7). Geben Sie nun aber als url

```
192.168.1.10/my_index.html
```

ein (roter Pfeil). Dann wird der Inhalt der html-Seite angezeigt und Sie können auch den Links der Textfiles (blauer Pfeil) folgen.



So lange die Applikation nicht gestartet wurde, liegen aber noch keine Bilder im

Verzeichnis und die Webseite ist ansonsten leer.

15. Nun wollen wir die Applikation starten. Da aber noch die Applikation `vcagrabber` (vgl. Abbildung 7) läuft, welche die Kamera blockiert, müssen wir diese zuerst stoppen. Dies erfolgt mit

```
sudo /etc/init.d/vcagrabber-init stop
```

Nun können Sie die Applikation im `tmp` Folder auf dem Raspberry Pi starten mittels:

```
sudo ./app
```

Das Präfix `sudo` ist notwendig, weil die Applikation sonst keine Schreibrechte im Verzeichnis `/var/www` besitzt.

Beobachten Sie das Verhalten der Webseite. Sie können den Refresh auch manuell ausführen.

16. Starten Sie zum Schluss die Default Applikation wieder mittels:

```
sudo /etc/init.d/vcagrabber-init start
```

Sie können die Funktionalität im Browser (Abbildung 7) verifizieren.

An diesem Beispiel ist gut zu erkennen, wie Webserver und Browser verwendet werden können, um Bild- und Textdaten an den Host zu übertragen. Allerdings funktioniert der Zugriff auf die Daten noch nicht synchronisiert mit der Applikation und wenn sowohl die Applikation als auch der Webserver simultan auf die Ressourcen (d.h. die Files) zugreifen kommt es sicherlich zu einer Race Condition. Daher müssen Webserver und Applikation noch geeignet synchronisiert werden. Dies wird dadurch bewerkstelligt, dass die Bildverarbeitungsroutinen der OpenCV in einer Web Applikation eingebettet werden, welche die Resultate direkt an den Browser liefern kann. Die Applikation `vcagrabber` funktioniert genau nach diesem Prinzip und in der folgenden Übung werden wir dies an einem konkreten Beispiel kennenlernen.

### 4.2.3. Übung 7: Applikation Template

In diesem Beispiel werden wir das sog. Applikation Template kennenlernen, mit welchem wir einfach Bildverarbeitungsroutinen für die Raspberry Pi Plattform erstellen und die Ergebnisse visualisieren können. Dieses besteht letztendlich aus dem ausführbaren Programm `image_proc_agent` und den hierzu notwendigen Libraries. Zusätzlich werden wir zwei neue Konzepte kennenlernen.

- Einerseits werden wir sehen, dass die per Skript gesteuerte Erstellung von Binärcode (unser Beispiel `cl.sh`) in der Praxis durch sog. Makefiles erfolgt. Diese stellen ein sehr mächtiges Werkzeug dar, sind aber in der Syntax etwas kryptisch. Trotzdem werden wir mit unserem Vorwissen die grundlegenden Konzepte verstehen können und sollten zumindest in der Lage sein, bereits existierende Makefiles an die eigenen Bedürfnisse anzupassen (was meist auch in der Praxis ausreicht).

- Weiterhin werden wir in Form von Eclipse die Möglichkeit einer GUI-basierten Entwicklungsumgebung kennenlernen.

Ziele:

1. Wir kennen die wesentlichen Komponenten des Applikation Templates und wissen, an welcher Stelle individuelle Bildverarbeitungsroutrinen eingefügt werden können.
2. Wir können ein Makefile aufrufen und auf Basis der Targets ein Projekt bearbeiten.
3. Wir können ein existierendes Projekt in Eclipse importieren, die Files bearbeiten und die verschiedenen Targets des Makefiles ausführen.

Übung im Detail:

1. Wechseln Sie in das Verzeichnis `~/image_proc_agent` unter `/home/pi`. Dort liegen die notwendigen Quelldateien und Libraries zur Erstellung des Applikation Templates:

- a. Folder `externals`:

```
gd_lib
jpeg_lib
libudev
OpenCV3.0.0.linux.x86
openssl.linux.arm
openssl.linux.x86
vca_agent_lib
vca_agent_lib_teach
VCAHyperblock_lib
VCAParamHandler_lib
VCAProcess_lib
VCAUtils_lib
```

Hier liegen alle Libraries, welche zur Erstellung der Applikation `image_proc_agent` benötigt werden. Unter anderem finden sich hier die OpenCV Libraries für die x86-Host sowie die arm-Target Plattform.

- b. Die verschiedenen Quelldateien:

```
main.cpp
Process_2_Teach.cpp
Process_2_Teach.h
ProcessingDbgImages.cpp
ProcessingDbgImages.h
ProcessingRoiData.h
ProcessingTeachImageData.h
ProcessingTeachMain.cpp
ProcessingTeachMain.h
ProcessingTeachParams.cpp
ProcessingTeachParams.h
```

Dabei sind fürs erste nur die Dateien `Process_2_Teach.cpp` und `Process_2_Teach.h` wichtig. Dort wird der Code für die Bildverarbeitung eingefügt und zwar in der Methode (ab Zeile 46):

```
VcaProcBaseData* Process_2_Teach::processStep(...)
```

Hierzu kommen wir gleich nochmals.

c. Makefiles:

Die Files `Makefile` und `MakeOpenCV.mk` dienen der Erstellung des Programms.

d. Deployment Skript:

Das File `deployRaspiArm.sh` dient dem Deployment (d.h. dem Transfer) des arm Binaries auf das Raspberry Pi Target.

e. Konfigurationsfiles:

ebvOption.conf ebvOptionLocal.conf

Diese Files dienen der Konfiguration des Applikation Templates und werden beim Start als Kommandozeilenargument übergeben.

2. Erstellen Sie nun die Applikation, indem Sie

make

Auf der Kommandozeile ausführen. Make ist ein sehr mächtiges Build Management Tool, welches zur Erstellung von SW unter Linux verwendet wird. Lesen Sie kurz die „Quick Reference“ dazu in Kapitel 5.1.

- Studieren Sie die Ausgabe von `make` auf der Kommandozeile und versuchen Sie, in groben Zügen den `make` Prozess nachzuvollziehen. Überlegen Sie insbesondere, ob die Applikation für den Host oder das Raspberry Pi erstellt wird.

```
pi@raspi-SDK-VirtualBox:~/image_proc_agent$ make
mkdir -p linux-obj_rel
g++ -I ./externals/vca_agent_lib/include -I ./externals/VCAParamHandler_lib/include -I ./externals/VCAUtlis_lib/include -I ./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/include/opencv2 -DNO_SSL_DL -DOSC_HOST -fpic -c linux-obj_rel
g++ -I ./externals/vca_agent_lib/include -I ./externals/VCAParamHandler_lib/include -I ./externals/VCAUtlis_lib/include -I ./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/include/opencv2 -DNO_SSL_DL -DOSC_HOST -fpic -o linux-obj_rel
g++ -I ./externals/vca_agent_lib/include -I ./externals/VCAParamHandler_lib/include -I ./externals/VCAUtlis_lib/include -I ./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/include/opencv2 -DNO_SSL_DL -DOSC_HOST -fpic -c linux-obj_rel
g++ -I ./externals/vca_agent_lib/include -I ./externals/VCAParamHandler_lib/include -I ./externals/VCAUtlis_lib/include -I ./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/include/opencv2 -DNO_SSL_DL -DOSC_HOST -fpic -o linux-obj_rel
g++ -I ./externals/vca_agent_lib/include -I ./externals/VCAParamHandler_lib/include -I ./externals/VCAUtlis_lib/include -I ./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/include/opencv2 -DNO_SSL_DL -DOSC_HOST -fpic -c linux-obj_rel
echo target: ..
target: ..
echo g++ -L./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/lib -L./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/lib/linux-obj_rel -L./externals/vca_agent_lib/linux-obj_rel -L./externals/VCAHyperblock_lib/linux-obj_rel -L./externals/vca_agent_lib/teach/linux-obj_rel -L./externals/openssl.linux.x86/lib/opensslTeachParams.o linux-obj_rel/main.o -lvca agent lib teach -lVCAParamHandler -lVCAProcess lib -lvca agent vncvideostab -lopencv_calib3d -lopencv_features2d -lopencv_highgui -lopencv_videoio -lopencv_imgcodecs -lopencv_imgproc -llibwebp -llibtiff -llibjasper -lllmfmmf -ldl -lrt -lm -lpthread -lsdl -lcrypto -lzlib -llibudev.so.1 -o i
g++ -L./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/lib -L./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/lib/linux-obj_rel -L./externals/vca_agent_lib/linux-obj_rel -L./externals/VCAHyperblock_lib/linux-obj_rel -L./externals/vca_agent_lib/teach/linux-obj_rel -L./externals/openssl.linux.x86/lib/opensslTeachParams.o linux-obj_rel/main.o -lvca agent lib teach -lVCAParamHandler -lVCAProcess lib -lvca agent vncvideostab -lopencv_calib3d -lopencv_features2d -lopencv_highgui -lopencv_videoio -lopencv_imgcodecs -lopencv_imgproc -llibwebp -llibtiff -llibjasper -lllmfmmf -ldl -lrt -lm -lpthread -lsdl -lcrypto -lzlib -llibudev.so.1 -o i
g++ -L./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/lib -L./externals/OpenCV3.0.0.linux.x86/build/linux-obj_rel/lib/linux-obj_rel -L./externals/vca_agent_lib/linux-obj_rel -L./externals/VCAHyperblock_lib/linux-obj_rel -L./externals/vca_agent_lib/teach/linux-obj_rel -L./externals/openssl.linux.x86/lib/opensslTeachParams.o linux-obj_rel/main.o -o linux-obj_rel/image_proc_agent -lvca agent lib teach -lVCAParamHandler -lVCAProcess lib -lvca agent vncvideostab -lopencv_calib3d -lopencv_features2d -lopencv_highgui -lopencv_videoio -lopencv_imgcodecs -llibjpeg -llibwebp -llibtiff -llibjasper -lllmfmmf -ldl -lrt -lm -lpthread -lsdl -lcrypto -lzlib -llibudev.so.1
pi@raspi-SDK-VirtualBox:~/image_proc_agent$ ls -l
```

4. Studieren Sie das Makefile und finden Sie die Zeilen

```
ifeq ($(TARGET_TYPE),arm)
```

```
CXX=arm-linux-gnueabi-g++
```

sowie etwas weiter:

```
else  
  CXX      = g++
```

Mit diesen Anweisungen wird der Compiler bzw. Cross Compiler ausgewählt.  
Mit der Anweisung

```
make
```

wird der else-Zweig ausgeführt und das Applikation Template für den Host erstellt. Das Binärfile liegt im Unterverzeichnis

```
linux-obj_rel
```

Der Vorteil einer Erstellung für den Host ist, dass eine ggf. notwendige Fehlersuche vereinfacht wird.

Um aber den if-Zweig auszuführen und das Applikation Template für das Raspberry Pi Target zu erstellen ist die folgende Anweisung auszuführen:

```
TARGET_TYPE=arm make
```

Das zugehörige Binärfile liegt dann im Unterverzeichnis

```
arm-obj_rel
```

Führen Sie dies aus, verifizieren Sie die Erstellung des Binärfiles und testen Sie, dass die Ausführung auf dem Host fehlschlägt.

##### 5. Nun übertragen wir das Binärfile mit dem Skript

```
deployRaspiArm.sh
```

auf den Raspberry Pi. Öffnen Sie das File und versuchen Sie in groben Zügen die Funktionsweise nachzuvollziehen.

Finden Sie z.B. Zeilen mit den Ihnen bereits bekannten `ssh` bzw. `scp` Statements. Dabei werden Sie feststellen, dass jeweils das Statement

```
sshpass -p $TARGET_PASS ...
```

den `ssh` bzw. `scp` Statements vorausgehen. Dies, damit das Passwort für den Verbindungsaufbau (in der Variablen `TARGET_PASS`) nicht jedes Mal eingegeben werden muss<sup>30</sup>.

Führen Sie nun das Skript aus mit

```
./deployRaspiArm.sh
```

---

<sup>30</sup> Dies birgt natürlich ein gewisses Sicherheitsrisiko und sollte daher für produktive Anwendungen nicht verwendet werden.

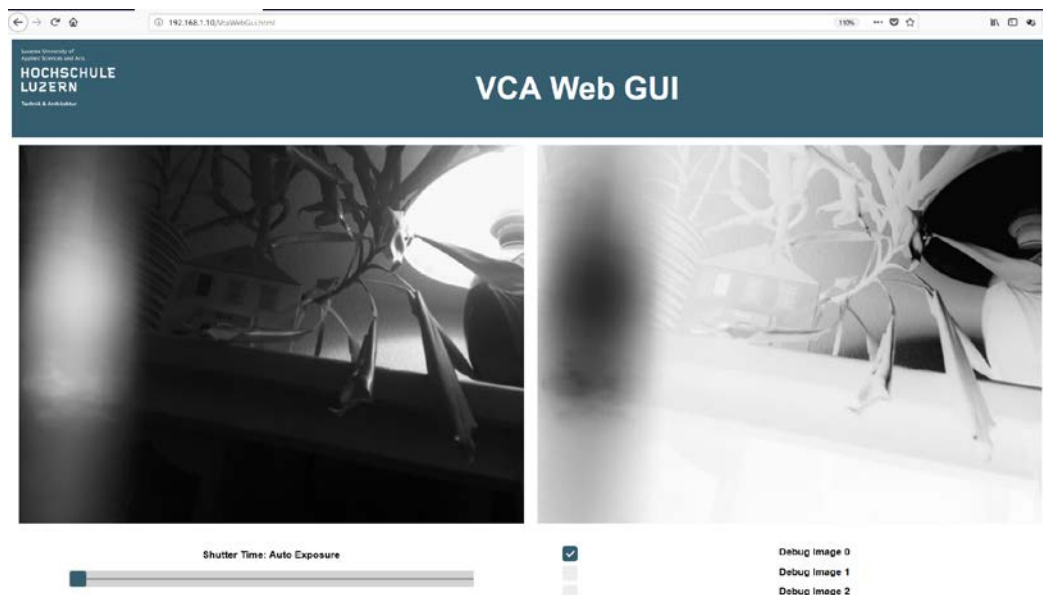
Auf der Kommandozeile erscheint dann (etwas zeitverzögert) die Ausgabe des Applikation Templates

```

httpserver.passphrase=
ivpserver.port=29001
diag.loglevel=1
diag.verbosity=1
diag.numsteps=-1
img.subsample=1
img.cropx=0
img.cropy=0
img.aliveRate=0
img.jpgQuality=-1
***** {0}
04/09 18:41:50.495 INF MainProcess:classFactory() called.
04/09 18:41:50.495 INF MainProcess:registerProcess: Process2Teach [0]
04/09 18:41:50.495 INF MainProcess:initialize: Process2Teach
Registered processes:
  0: Process2Teach
    {0}
04/09 18:41:50.499 INF HTTP server started
04/09 18:41:50.499 INF try connect 127.0.0.1:29000
04/09 18:41:50.504 INF IVP connect succeeded
04/09 18:41:50.530 INF 0_Process2Teach: Calculate inverse image.
04/09 18:41:50.680 INF 0_Process2Teach: Calculate inverse image.
04/09 18:41:50.824 INF 0_Process2Teach: Calculate inverse image.
04/09 18:41:50.974 INF 0_Process2Teach: Calculate inverse image.
04/09 18:41:51.147 INF 0_Process2Teach: Calculate inverse image.
04/09 18:41:51.301 INF 0_Process2Teach: Calculate inverse image.
04/09 18:41:51.454 INF 0_Process2Teach: Calculate inverse image.
04/09 18:41:51.602 INF 0_Process2Teach: Calculate inverse image.

```

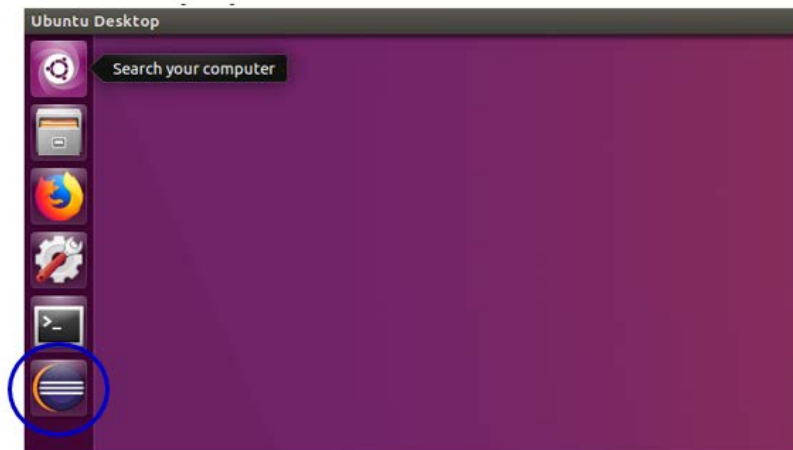
Beobachten Sie die Ausgabe im Browser Fenster (vgl. Abbildung 7). Nun wird im rechten Teil des Browsers das Ergebnis der Bildverarbeitungskette der Applikation angezeigt. Dabei wird aktuell nur das Debug Image 0 verwendet, die beiden anderen sind leer.



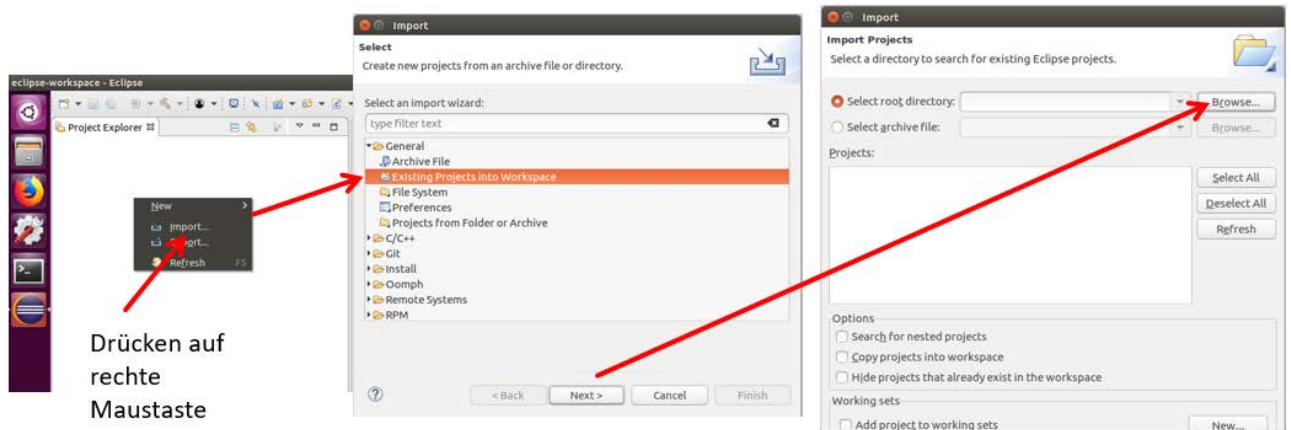
Damit beherrschen wir den Build- und Deploy-Prozess für die Verwendung des Applikation Templates von der Kommandozeile aus. Damit können wir uns dem Eclipse IDE zuwenden, welches die gleiche Funktionalität noch etwas komfortabler anbietet.



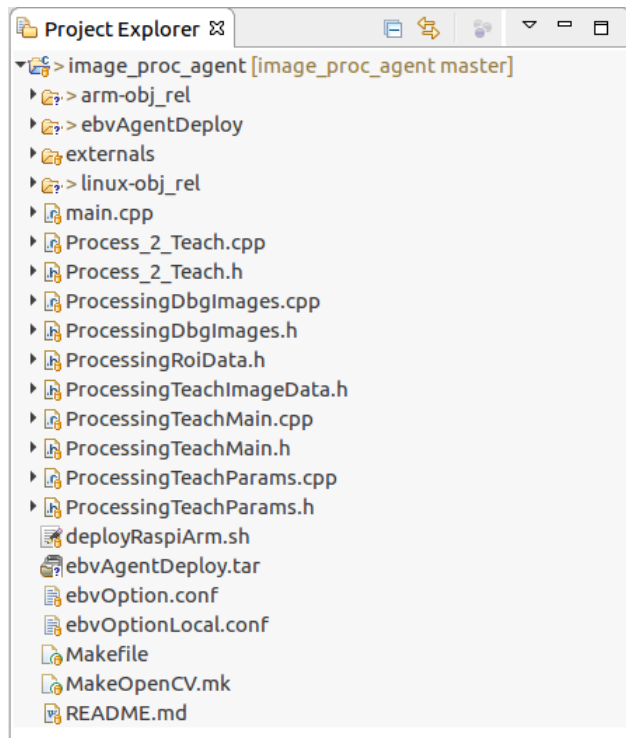
6. Öffnen Sie das Eclipse IDE<sup>31</sup> durch Drücken des entsprechenden Buttons auf dem VM Desktop.



7. Importieren Sie das Projekt `image_proc_agent`:



<sup>31</sup> Integrated Development Environment



8. Machen Sie sich ein wenig mit dem Eclipse Tool vertraut, indem Sie durch den Source Code Browsen. Identifizieren Sie im File `Process_2_Teach.cpp` in Zeile 65 z.B. das Statement:

```
mProcImage = cv::Mat(inputImage.rows, inputImage.cols,
                      inputImage.type());
```

Hier wird das Eingabebild (`inputImage`) durch Aufruf des Konstruktors der Klasse `cv::Mat` in die Variable `mProcImage` geschrieben<sup>32</sup>. Mit Hilfe der Klasse `cv::Mat` werden in OpenCV allgemein Matrizen und insbesondere Bilder repräsentiert<sup>33</sup>.

In den folgenden Zeilen werden die Pixel des Bildes `mProcImage` dann Zeilen- und Spalten-weise verändert:

```
for (int rows = 0; rows < inputImage.rows; rows++) {
    for (int cols = 0; cols < inputImage.cols; cols++) {
        mProcImage.at<unsigned char>(rows, cols) = 255 -
            inputImage.at<unsigned char>(rows, cols);
    }
}
```

Dies um zu illustrieren, wie Pixel einzeln adressiert werden können. Meist gibt es aber in OpenCV direkte geeignete Funktionen, um die Operation mit einem Aufruf durchzuführen. Hier kann das Inverse einfach wie folgt bestimmt werden:

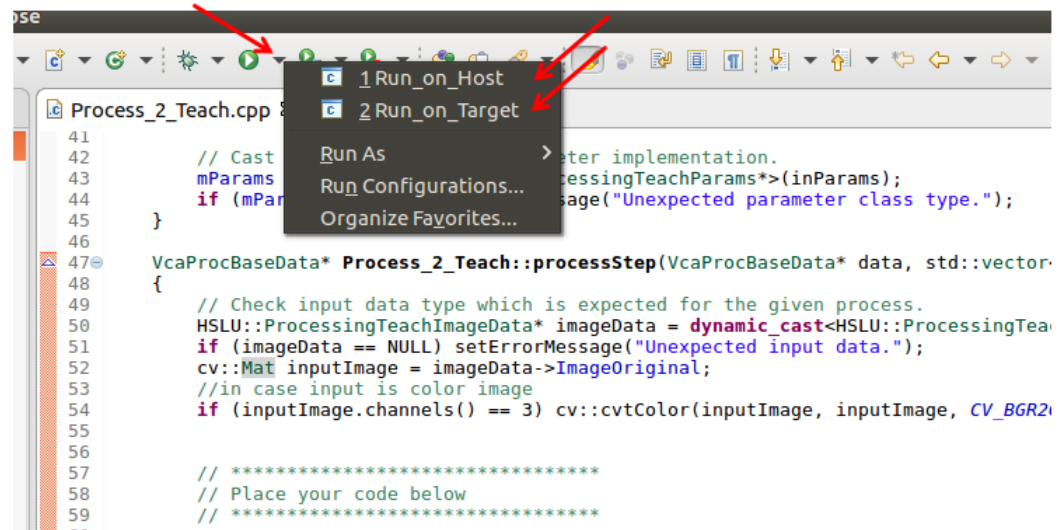
```
cv::subtract(255, inputImage, mProcImage);
```

<sup>32</sup> Hier dient dieses Statement nur dazu, das Bild `mProcImage` auf die identische Grösse und Typ wie `inputImage` zu initialisieren.

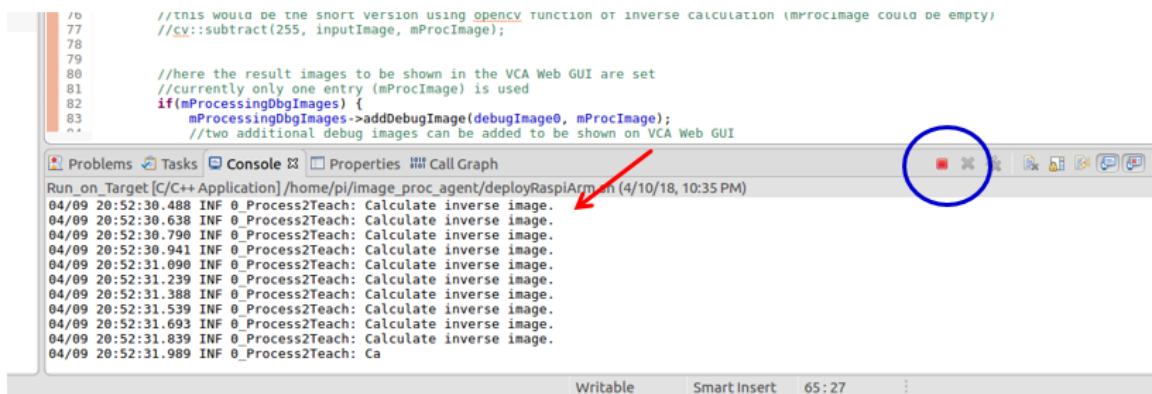
<sup>33</sup> [https://docs.opencv.org/3.0-beta/modules/core/doc/basic\\_structures.html#mat](https://docs.opencv.org/3.0-beta/modules/core/doc/basic_structures.html#mat)

Diese Zeile ist aktuell kommentiert<sup>34</sup>.

- Sie können nun die Eclipse Umgebung direkt für die Erstellung und das Deployment der Applikation verwenden. Hierzu auf den schwarzen Pfeil neben dem grünen «Run» Button drücken. Damit öffnet sich das Run-Context-Menü. Sie können zwischen Run\_on\_Host und Run\_on\_Target wählen.



Damit sollte die Applikation laufen. Dies sehen Sie auch an der Ausgabe im Konsolenfenster (unten). Zum Stoppen der Applikation auf den roten Stopp Button im Konsolenfenster drücken.



Damit haben wir den ersten wesentlichen Meilenstein erreicht:

- In Form des Applikation Templates haben wir ein Mittel an der Hand, um Bildverarbeitungsalgorithmen live zu testen (der Code ist im File `Process_2_Teach.cpp` zu integrieren) und per Browser in Echtzeit anzuzeigen.

<sup>34</sup> Bei Verwendung dieser Funktion ist es nicht nötig, vorab das Bild `mProcImage` auf die identische Grösse und Typ wie `inputImage` zu initialisieren. Dies macht die OpenCV Funktion selbst.

- Mit Hilfe des Eclipse IDE verfügen wir über eine Entwicklungsumgebung, die ein einfaches Browsen durch den Source Code ermöglicht und die Erstellung mittels Makefiles integriert hat.
- Wir haben ein erstes grobes Verständnis der Zusammenhänge beim Kompilieren und Linken der Anwendung und können diese sukzessive ausbauen.
- Wir haben in Form des OpenCV Frameworks ein mächtiges Tool an der Hand, um sukzessive komplexe Bildverarbeitungsroutinen für das Raspberry Pi Target zu erstellen.

## 5. Eine Einführung in Makefiles

### 5.1.Quick Reference

Make ist ein sehr mächtiges Build Management Tool, welches zur Erstellung von SW unter Linux verwendet wird. Es arbeitet mit sog. Targets, die nach definierten Regeln („Rules“) erstellt werden. Die Regeln sind in einem Makefile<sup>35</sup> abgelegt, welches bei der Ausführung von `make` gelesen wird. Die zwei Targets im Makefile des Applikation Templates (s. Übung 7, 4.2.3) sind:

- `all`:  
Default Target, welches das gesamte Projekt erstellt.
- `clean`:  
Löscht alle Objekt- und Binärfiles.

Hierbei

Die Syntax zum Aufruf von `make` ist (z.B. für das Target `clean`):

- Falls das Makefile auch wirklich `Makefile` heisst und im aktuellen Verzeichnis liegt:

```
make clean
```

- Falls das Makefile z.B. `Makefile.bf` heisst und im aktuellen Verzeichnis liegt:

```
make -f Makefile.bf clean
```

- Falls das Default Target (an oberster Stelle im Makefile) aufgerufen werden soll:

```
make
```

### 5.2.Erweiterte Einführung in Makefiles

<sup>36</sup>Die folgenden Abschnitte soll eine Einführung in das Erstellen von Makefiles geben. Sie sind für Personen gedacht, die wissen, was das Programm `make` macht, und wozu man es verwendet, selber aber noch nie ein `makefile` erstellt haben. Bei den Beispielen liegt der Schwerpunkt auf Makefiles für C-Programme. Es gibt mehrere `make`-Programme, deren Makefiles nicht völlig kompatibel sind. Die folgenden Seiten beziehen sich auf das GNUmake. Speziell alles, was mit Pattern zu tun hat, kann je nach verwendetem `make`-Programm leicht unterschiedlich sein.

- [Targets, Regeln und Abhängigkeiten](#)
- [Das Defaulttarget](#)
- [Pattern in Regeln](#)
- [Suffix-Regeln](#)

---

<sup>35</sup> Makefile ist hier als Platzhalter für irgendein File mit Rules für `make` verwendet. Meist wird aber der Name auch direkt als Filename verwendet.

<sup>36</sup> Die folgenden Seiten wurden im Wesentlichen „as is“ von [14] übernommen, um einige Grundlagen zu Makefiles zum direkten Nachschlagen zur Verfügung zu haben.

- [Variablen in Makefiles](#)
  - [Kommentare in Makefiles](#)
  - [Phony targets](#)
  - [Pattern Substitution](#)
  - [Abhängigkeiten als Target \(\[make dep\]\(#\)\)](#)
  - [Rekursives Make](#)
  - [Typische Probleme mit make](#)
  - [Weitere Informationen](#)
- 

## 5.3.Targets, Regeln und Abhängigkeiten

Ein Makefile ist dazu da, dem Programm `make` mitzuteilen, was es tun soll (dies ist das "Target"), und wie es es tun soll (dies ist die zum Target gehörige "Regel"). Des Weiteren kann man zu jedem Target angeben, von welchen anderen Targets oder Dateien dieses abhängt.

Am besten sieht man das an einem einfachen Beispiel. Nehmen wir an, dass wir ein kleines C-Programm namens `prog.c` mit zugehöriger Header-Datei `prog.h` haben, welches wir normalerweise mit

```
gcc -o prog prog.c
```

übersetzen. Unser Ziel ist es also, die Datei `prog` zu erzeugen. Diese ist offensichtlich abhängig von `prog.c` und `prog.h`. Im Makefile sieht dies wie folgt aus:

```
prog: prog.c prog.h
    gcc -o prog prog.c
```

In der ersten Zeile teilen wir `make` mit, dass es ein Target namens "`prog`" gibt, und dass dieses von `prog.c` und `prog.h` abhängt. In der zweiten Zeile sagen wir `make`, mit welcher Regel (d.h. wie) es dieses Target erzeugen kann. Wird `make` nun aufgerufen, überprüft es, ob eine der beiden Dateien `prog.c` oder `prog.h` neuer ist als das Target. In diesem Fall führt es die zweite Zeile aus und erzeugt eine neue ausführbare Datei.

**Eine gemeine Falle für Anfänger ist, dass die zweite Zeile mit einem <tab> anfangen muss, und nicht mit Leerzeichen.**

Viele Targets können nicht wie hier durch einen einzigen Befehl erzeugt werden, sondern es sind mehrere nötig. In diesem Fall folgen auf die Zeile mit den Abhängigkeiten einfach mehrere Zeilen, die alle mit <tab> anfangen. Auch die Abhängigkeiten für ein Target dürfen auf mehrere Zeilen verteilt sein.

Man beachte, dass in unserem Beispiel "`prog`" gleichzeitig ein Name für ein Target und für eine Datei ist. `make` sieht darin keinen Unterschied. Auch die beiden Dateien `prog.c` und `prog.h` sind für `make` nichts anderes als Targets. Diese Targets hängen von nichts ab, sind also immer aktuell, und es gibt auch keine Regel, um sie zu erzeugen. Würde nun beispielsweise die Datei `prog.h` nicht existieren, so würde `make` feststellen, dass es das Target `prog.h`, welches für `prog` benötigt wird, nicht erzeugen kann. Die Fehlermeldung lautet dementsprechend:

```
make: *** No rule to make target `prog.h'. Stop.
```

Die Zeile mit der Regel (`gcc ...`) wird von `make` in einer shell aufgerufen. Es ist also möglich, wildcards oder Kommandosubstitutionen (mit ``...``) zu benutzen.

---

## 5.4. Das Default Target

Beim Aufruf von `make` kann man das zu erzeugende Target angeben:

```
make prog
```

Ruft man `make` dagegen ohne jegliche Argumente auf, so wird das erste Target, welches im Makefile gefunden wird, erzeugt. In vielen Makefiles ist das erste Target deshalb eines namens `all`, welches von einer ganzen Reihe von weiteren Targets abhängt. Ziel ist es in der Regel, durch einen argumentlosen Aufruf von `make` ein fertiges, ausführbares Programm zu bekommen. Insbesondere müssen hierzu alle Objekdateien und die ausführbaren Dateien erzeugt werden.

## 5.5. Pattern in Regeln

In der Praxis bestehen Programmierprojekte aus so vielen Dateien und Abhängigkeiten, dass es impraktikabel ist, für jede einzelne Objekt-Datei eine neue Regel ins Makefile einzubauen, und diese bei jedem neuen `#include` zu ändern. Deshalb gibt es die Möglichkeit, Regeln durch eine Art Wildcard-Pattern zu definieren:

```
%.o: %.c
    gcc -Wall -g -c $<
```

Diese Regel besagt, dass jede `.o`-Datei von der entsprechenden `.c`-Datei abhängt, und wie sie mit Hilfe des Compilers erzeugt werden kann.

Um innerhalb der Regel auf den Namen des Targets und die der Abhängigkeiten zugreifen zu können, gibt es einige von `make` vordefinierte Variablen. Hier die meiner Meinung nach wichtigsten (von sehr vielen):

<code>\$&lt;</code>	die erste Abhängigkeit
<code>\$@</code>	Name des targets
<code>\$+</code>	eine Liste aller Abhängigkeiten
<code>\$^</code>	eine Liste aller Abhängigkeiten, wobei allerdings doppelt vorkommende Abhängigkeiten eliminiert wurden.

Man beachte, dass bei der Angabe der Abhängigkeiten hier die Abhängigkeit der Objekt-Dateien von diversen Header-Dateien unter den Tisch gefallen ist. Eine Möglichkeit, diese wiederzubekommen, werde ich im [Abschnitt über Abhängigkeiten als target](#) besprechen.

Die hier vorgestellte Art, durch Pattern Regeln zu erzeugen, ist nur eine von vielen möglichen. Leider muss man sagen, dass Pattern und Pattern-Substitutionen in Makefiles zu den Sachen gehören, die am meisten durcheinandergeraten und am meisten verwirren. Außerdem unterscheiden sie sich bei unterschiedlichen `make`-Versionen u.U. voneinander.

Regeln, die durch Pattern erzeugt wurden, sind ein Spezialfall sog. "Impliziter Regeln", d.h. Regeln, die man `make` nicht explizit angegeben hat, sondern die von `make` durch Anwendung bestimmter Vorschriften deduziert werden.



## 5.6. Variablen in Makefiles

Es ist möglich, in Makefiles Variablen zu definieren und zu benutzen. Üblicherweise verwendet man Großbuchstaben. Gebräuchlich sind beispielsweise folgende Variablen:

CC	Der Compiler
CFLAGS	Compiler-Optionen
LDFLAGS	Linker-Optionen

Auf den Inhalt dieser Variablen greift man dann mit `$(CC)`, `$(CFLAGS)` bzw. `$(LDFLAGS)` zurück.

Ein einfaches Makefile eines Programmes namens `prog`, welches aus einer Reihe von Objekt-Dateien zusammengelinkt werden soll, könnte also wie folgt aussehen:

```
VERSION = 3.02
CC       = /usr/bin/gcc
CFLAGS   = -Wall -g -D_REENTRANT
LDFLAGS  = -lm -lpthread

OBJ = datei1.o datei2.o datei3.o datei4.o
datei5.o

prog: $(OBJ)
    $(CC) $(CFLAGS) -o prog $(OBJ)
    $(LDFLAGS)

%.o: %.c
    $(CC) $(CFLAGS) -c $<
```

Das Default Target ist hier das ausführbare Programm `prog`. Dieses hängt von allen Objekt-Dateien ab. Beim Linken werden die `math` und die `pthread` Library dazugelinkt.

## 5.7. Kommentare in Makefiles

Genau wie die Shell werden von `make` Zeilen, die mit einem `"#"` anfangen, als Kommentare angesehen:  
`#` auskommentierte Zeile.

## 5.8. Phony targets

Bei normalen Targets überprüft `make`, ob sie aktueller sind als alle Targets, von denen sie abhängen. Falls dies nicht der Fall ist, werden sie neu erzeugt. Targets, die von keinen weiteren Targets abhängen, sind folglich immer aktuell und werden nie erzeugt. Dies trifft in der Regel beispielsweise für die Quellen eines Programmes zu.

Manche Targets entsprechen keiner physikalischen Datei. Ein typisches solches Target ist das Target `"clean"`, welches alle erzeugten Dateien löscht. Es ist von nichts abhängig. Probleme kann es nun geben, wenn es im Verzeichnis eine Datei namens `"clean"` gibt. `Make` stellt nun fest, dass das Target `"clean"` bereits erzeugt ist, und von nichts abhängig und deshalb aktuell ist.

Es gibt also Targets, die stets(!) neu erzeugt werden sollen, unabhängig davon, von welchen anderen Targets sie abhängig sind. Diese nennt man phony Targets:

```
OBJ = datei1.o datei2.o datei3.o datei4.o
datei5.o
BIN = prog

.PHONY: clean
clean:
    rm -rf $(BIN) $(OBJ)
```

Gibt man "make clean" ein, so wird nun der `rm`-Befehl stets durchgeführt, unabhängig davon, ob eine Datei namens "clean" existiert oder nicht. Phony Targets können genau wie alle normale Targets von anderen abhängen. Im Allgemeinen ist es nicht nötig, irgendwelche Targets als phony zu deklarieren. Man sollte das ganze jedoch im Hinterkopf behalten.

## 5.9. Pattern Substitution

Genau, wie man mit Hilfe von Pattern Regeln erzeugen konnte, kann man auch Variablen erzeugen:

```
OBJ = datei1.o datei2.o datei3.o datei4.o
datei5.o
SRC = $(OBJ:%.o=%.c)
HDR = $(OBJ:%.o=%.h) config.h
```

Hier haben die Variablen `SRC` und `HDR` danach die Werte

```
datei1.c datei2.c datei3.c datei4.c datei5.c
datei1.h datei2.h datei3.h datei4.h datei5.h config.h
```

Genau wie bei der Erzeugung von Regeln mit Hilfe von Pattern gilt auch hier, dass die Möglichkeiten, Variablen durch solche Substitutionen zu erzeugen, so mannigfaltig sind, dass man sie eigentlich nicht alle kennen kann. Im Zweifelsfall empfiehlt sich, die `make`-Anleitung durchzulesen.

## 5.10. Abhängigkeiten als Target (`make dep`)

Benutzt man implizite oder durch Pattern erzeugte Regeln zur Erzeugung von Objekt-Dateien, so entfallen die Abhängigkeiten dieser von den Header-Dateien. Um dieses Problem zu umgehen, ohne die Abhängigkeiten jeder Objekt-Datei per Hand ins Makefile einbauen zu müssen, definiert man meist ein Target namens "dep" für "dependencies":

```
SRC = datei1.c datei2.c datei3.c datei4.c datei5.c
CC  = /usr/bin/gcc
DEPENDFILE = .depend

dep: $(SRC)
    $(CC) -MM $(SRC) > $(DEPENDFILE)

-include $(DEPENDFILE)
```

Die Option `-MM` lässt den Präcompiler nach `#include`-Direktiven im Quellcode schauen. Er gibt die entsprechenden Abhängigkeiten genau in dem Format, in dem sie `make` braucht, auf der Standardausgabe aus. Hier werden sie allerdings in eine Datei namens `.depend` umgelenkt, welche dann mittels eines `include` in das Makefile eingefügt wird. Das Minuszeichen vor dem `include` sagt `make`, dass es nicht mit einer Fehlermeldung abbrechen soll, wenn die Datei nicht existiert - schließlich muss sie ja erst einmal durch einen Aufruf von `"make dep"` erzeugt werden. (Der `include`-Befehl ist eine Spezialität des GNUmake-Programmes. Wie man eine entsprechende Wirkung bei anderen `make`-Programmen erzielt, weiß ich nicht.)

Ein solches Target ist insbesondere dann wichtig, falls es Header-Dateien gibt, die dynamisch generiert werden und anschließend in Quell-Code-Dateien eingebunden werden.

## 5.11. Rekursives Make

Bei großen Projekten sind die Quelldateien über viele Verzeichnisse in mehreren Ebenen verstreut. Aus Gründen der Übersichtlichkeit und auch der Effektivität ist es in solchen Fällen üblich, jedem Verzeichnis sein eigenes Makefile zu geben. Man kann dann z.B. in einem Verzeichnis ein `"make clean"` machen, ohne dass danach der gesamte Verzeichnisbaum neu kompiliert werden muss.

Im Top-Level-Verzeichnis befindet sich dann nur noch ein Makefile, welches rekursiv `make` in allen Unterverzeichnissen aufruft. Des Weiteren könnte man dort eine Datei mit allgemeinen Regeln ablegen, die von allen Makefiles in den Unterverzeichnissen eingebunden wird.

Es gibt keine Standard-Methode, dies alles zu machen. Es hängt zu sehr von den besonderen Bedingungen, die beim Projekt herrschen, ab. Hier ein aus Gründen der Übersichtlichkeit äußerst rudimentär gehaltenes Beispiel:

```
# Makefile im Top-Level-Verzeichnis

DIRS = dir1 dir2 dir3

compile:
    for i in $(DIRS); do make -c $$i;
done
```

Das `$$i` wird von `make` zu `$i` evaluiert. Die Shell referenziert dann den Wert der Variablen `i`. Die Option `-c`, die dem `make` übergeben wird, bewirkt, dass `make` vor dem Start ins angegebene Verzeichnis wechselt.

```
# Makefile.rules im Top-Level-Verzeichnis

CC      = /usr/bin/gcc
CFLAGS  = -Wall -g

%.o:%.c
    $(CC) $(CFLAGS) -c $<
```

In dieser Datei werden Regeln und Variablen definiert, die in allen Unterverzeichnissen gelten sollen. Was noch fehlt, sind die Makefiles in den einzelnen Unterverzeichnissen:

```
# Makefile in einem Unterverzeichnis
include ../Makefile.rules
```

```
OBJ = datei1.o datei2.o datei3.o datei4.o  
datei5.o
```

```
all: $(OBJ)
```

Hier werden also nur noch die in diesem Verzeichnis zu erzeugenden Objekt-Dateien aufgelistet und das Default-Target so gesetzt, dass sie bei einem Aufruf von `make` ohne Argumente erzeugt werden.

## 6. Anwendungen

In diesem Kapitel werden wir verschiedene Bildverarbeitungsanwendungen für den Raspberry Pi in Form von Beispielprogrammen entwickeln.

### 6.1. Übung 8: Change Detection

Wir werden in diesem Beispiel die Anwendung aus Kapitel 2.4.1 und 2.4.2 des Theorie Skriptes [15] für die Raspberry Pi HW implementieren. Zusätzlich wollen wir im Rahmen dieser Übung den Umgang mit dem Versionierungstool `git` kennenlernen [10].

#### Ziele:

1. Wir können einfache Bildverarbeitungsalgorithmen auf dem Raspberry Pi implementieren.
2. Wir kennen die Grundlagen der Change Detektion.
3. Wir können Änderungen an unseren Programmen mittels des Versionierungstools `git` verfolgen und jederzeit zu einer älteren Version zurückkehren<sup>37</sup>.

#### Übung im Detail:

1. Öffnen Sie das Eclipse IDE und speziell das File `Process_2_Teach.cpp`. an der Stelle, an welcher der «produktive Code» eingefügt werden soll (s. Punkt 8 bzw. 9, Übung 7, ). Für die Change Detection Anwendung (Kapitel 2.4.1) benötigen wir die Differenz des aktuellen Bildes mit dem letzten Bild. Das aktuelle Bild ist mittels des Variable `inputImage` verfügbar. Um Zugriff auf des letzte Bild der Verarbeitung zu haben, müssen wir dies in einer neuen Membervariablen der Klasse `Process_2_Teach` speichern. Definieren Sie hierzu im File `Process_2_Teach.h` ein neues Member z.B. mit dem Namen `mPrevImage`. Um nun eine Kopie des aktuellen Bildes `inputImage` zu erstellen müssen Sie die folgende Befehlszeile verwenden:

```
mPrevImage = inputImage.clone();
```

#### **Hierbei ist folgender Punkt zu beachten:**

Würden Sie das Statement in der Form

```
mPrevImage = inputImage;
```

formulieren, dann würde die Variable `mPrevImage` nur eine Referenz auf `inputImage` halten und wenn sich letzteres ändert dann auch ersteres. Der Sinn hinter diesem Vorgehen ist, unnötiges Kopieren von Bildinhalten (z.B. bei der Übergabe an eine Funktion) zu vermeiden. Das heisst aber, dass Sie sich immer die Frage stellen müssen, ob das Bild wirklich verändert wird oder

---

<sup>37</sup> Sehen Sie hierzu auch den Annex, Kapitel 9.4.

konstant bleibt. Im Zweifelsfall eher wie oben die `clone()` Methode aufrufen.

Mit der Variablen `mPrevImage` haben wir nun also die Möglichkeit, auf das letzte Bild zuzugreifen. Da diese Variable aber beim allerersten Verarbeitungsschritt leer ist, bauen Sie die Verarbeitung wie folgt auf:

```
if (mPrevImage.size() != cv::Size()) {  
  
    ...Verarbeitung...  
  
}  
mPrevImage = inputImage;
```

D.h. beim ersten Mal wird der Verarbeitungsschritt übersprungen und nur am Ende das aktuelle Bild in `mPrevImage` gespeichert. Ab dem zweiten Schritt stehen dann immer beide Bilder für die Verarbeitung zur Verfügung.

2. Bauen Sie nun sukzessive die Change Detection an der oben markierten Stelle *...Verarbeitung...* auf:

- a. Differenzbild:

Für die Bestimmung des Differenzbildes verwenden Sie die Funktion `absdiff()`<sup>38</sup> z.B. wie folgt:

```
cv::Mat diffImage;  
cv::absdiff(inputImage, mPrevImage, diffImage);
```

Diese ermittelt den Absolut-Wert der Differenz von `inputImage` und `mPrevImage`. Dabei muss das Ergebnisbild `diffImage` nicht initialisiert sein, sondern wird bez. Grösse und Typ von der Funktion passend gefüllt.

- b. Binarisierung:

Für die Binarisierung verwenden wir die Funktion:

```
cv::Mat binaryImage;  
cv::threshold(diffImage, binaryImage, threshold, 255,  
              CV_THRESH_BINARY);
```

Diese binarisiert das Inputbild `diffImage` bezüglich des Schwellwertes `threshold` und schreibt das Ergebnis in die Variable `binaryImage`.

Hierbei können Sie den Schwellwert wie folgt definieren (auf eine Zeile!):

```
int32_t threshold = mParams->Process2Teach_binaryThreshold  
                    .getValueAsInt();
```

Damit können Sie den Wert dynamisch per Slider aus dem VCA Web GUI einstellen (vgl. Abbildung 7, rechts unten).

<sup>38</sup> Sie können Details zu den Funktionen immer in der online OpenCV Dokumentation nachlesen [8]. Trotzdem werden wir (zumindest am Anfang) die verwendeten Funktionen etwas detaillierter erläutern, um den Einstieg zu erleichtern.

- c. Morphologie:  
Morphologische Operationen auf dem Binärbild können folgendermassen durchgeführt werden:

```
cv::Mat kernel = cv::Mat::ones(5, 5, CV_8UC1);
cv::morphologyEx(binaryImage, binaryImage,
                  cv::MORPH_CLOSE, kernel);
```

Dabei wird mit der obigen Operation eine Schliessung auf dem Bild `binaryImage` mit dem vorab definierten Strukturelement `kernel` durchgeführt und das Ergebnis wieder direkt in `binaryImage` zurückgeschrieben. Alternative morphologische Operationen können Sie in der OpenCV Dokumentation nachschlagen [8].

- d. Region Labeling:  
Für die Ermittlung der zusammenhängenden Komponenten gehen Sie wie folgt vor:

```
cv::Mat stats, centroids, labelImage;
connectedComponentsWithStats(binaryImage, labelImage,
                             stats, centroids);
```

Hiermit werden ein Region Labeling auf dem Bild `binaryImage` durchgeführt und das Ergebnis in das Bild `labelImage` geschrieben. Dabei werden die folgenden Merkmale der Regionen ermittelt:

- Fläche
- Bounding Box
- Schwerpunkt

Diese sind in den `cv::Mat` Variablen `stats` und `centroids` abgelegt. Für den Zugriff gehen Sie wie folgt vor:

```
for (int i = 1; i < stats.rows; i++) {

    int topLeftx = stats.at<int>(i, 0);
    int topLefty = stats.at<int>(i, 1);
    int width = stats.at<int>(i, 2);
    int height = stats.at<int>(i, 3);

    int area = stats.at<int>(i, 4);

    double cx = centroids.at<double>(i, 0);
    double cy = centroids.at<double>(i, 1);

    ..do sth with values..
}
```

Es ist zu erkennen, dass die beiden Matrizen `stats` und `centroids` beide die gleiche Anzahl Zeilen hat (`rows`). Dabei hält die erste Zeile (Index 0) jeweils die Informationen des Hintergrunds. Daher startet der Loop erst mit Index 1.

Die Matrix `stats` hat 5 Spalten. Die ersten 4 Einträge bestimmen die



Bounding Box (`topLeftx/y`, `width/height`) und der fünfte die Anzahl Pixel (`area`). Alle Werte sind vom Typ `int`.

Die Matrix `centroids` hat 2 Spalten. Die beiden Einträge bestimmen den Schwerpunkt (`cx/cy`) und sind vom Typ `double`.

3. Für die Visualisierung des Ergebnisses haben wir drei Ausgabebilder zur Verfügung, welche im Browser angezeigt werden können (vgl. Punkt 5 in Übung 7). Diese werden an der folgenden Stelle zugewiesen:

```
if (mProcessingDbgImages) {
    mProcessingDbgImages->addDebugImage(debugImage0, mProcImage);
    ...
};
```

Dabei wird aktuell nur das `debugImage0` verwendet, die beiden anderen Bilder (`debugImage1/2`) sind auskommentiert. Hier können Sie nun die Variable `mProcImage` z.B. mit `labelImage` und die ??? (entsprechende Zeile im Code aktivieren) mit `binaryImage` ersetzen.

Zusätzlich erstellen wir noch ein Bild mit den Bounding Boxen und den Schwerpunkten. Hierzu machen wir eine Kopie des Inputbildes

```
cv::Mat resultImage = inputImage.clone();
```

und zeichnen die Ergebnisse der beiden Matrizen `stats` und `centroids` ein. Hierzu verwenden wir Rechtecke bzw. Kreise:

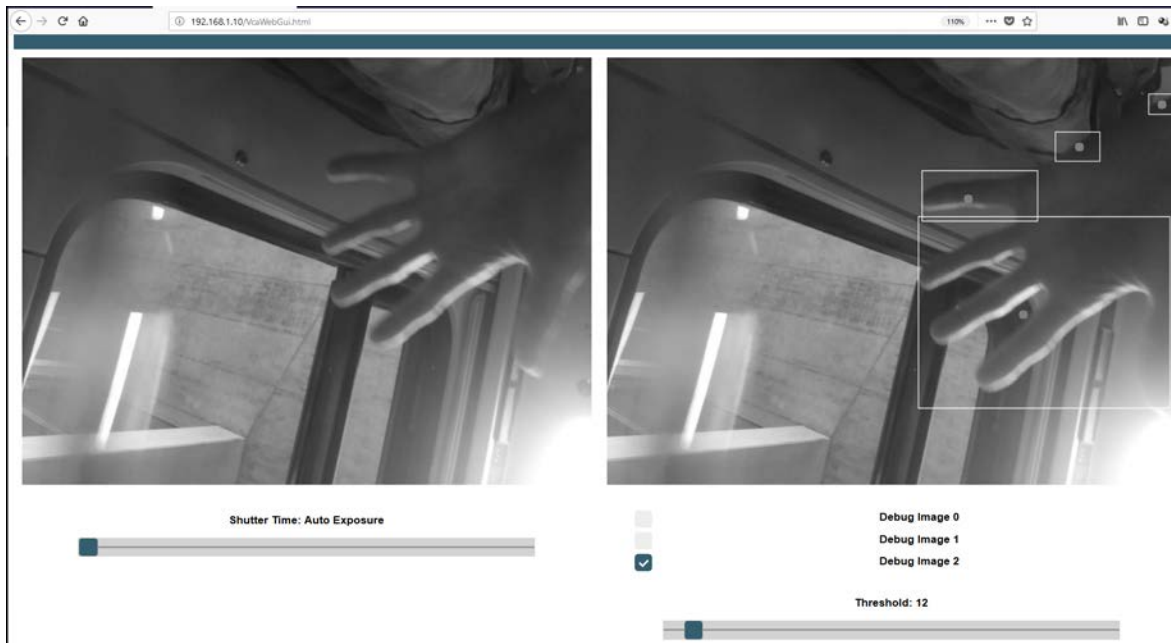
```
cv::Rect rect(topLeftx, topLeftty, width, height);
cv::rectangle(resultImage, rect, cv::Scalar(255, 0, 0));

cv::Point2d cent(cx, cy);
cv::circle(resultImage, cent, 5, cv::Scalar(128, 0, 0), -1);
```

Die beiden ersten Zeilen zeichnen ein Rechteck (`rect`) der angegebenen Grösse mit der Farbe `cv::Scalar(255, 0, 0)` in das Bild `resultImage`. Da `resultImage` nur ein Grauskalenbild ist, wird die Farbangabe als Weiss interpretiert<sup>39</sup>. Weiter Rechteckoptionen finden Sie in der online Dokumentation. Die beiden folgenden Zeilen zeichnen einen gefüllten Kreis (`thickness=-1`) mit Radius 5 an die Position `cx, cy`.

Fügen Sie nun das Bild `resultImage` noch als drittes Ausgabebild an und kontrollieren Sie das Ergebnis im Browser (s. Abbildung auf folgender Seite). Beachten Sie dabei, dass der Browser beim Start den Schwellwert `threshold` auf null setzt und Sie den Wert ggf. erhöhen sollten.

<sup>39</sup> Wir werden später noch sehen, dass für ein Farbbild ein rotes Rechteck resultieren würde.



4. Nach dem erfolgreichen Implementieren der Change Detektion wechseln Sie in ein Terminal Fenster und navigieren in das Verzeichnis

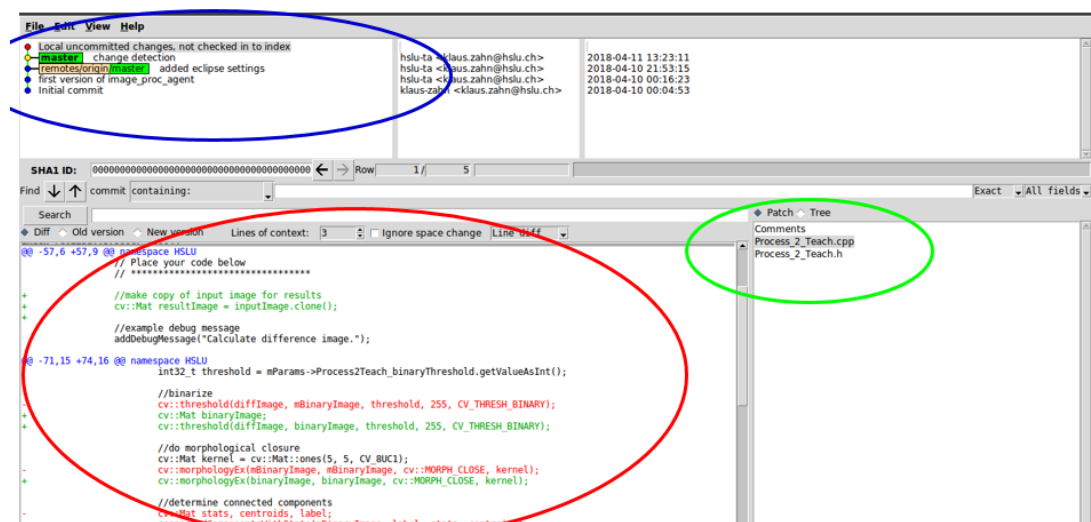
```
~/image_proc_agent
```

Führen Sie auf der Kommandozeile den Befehl

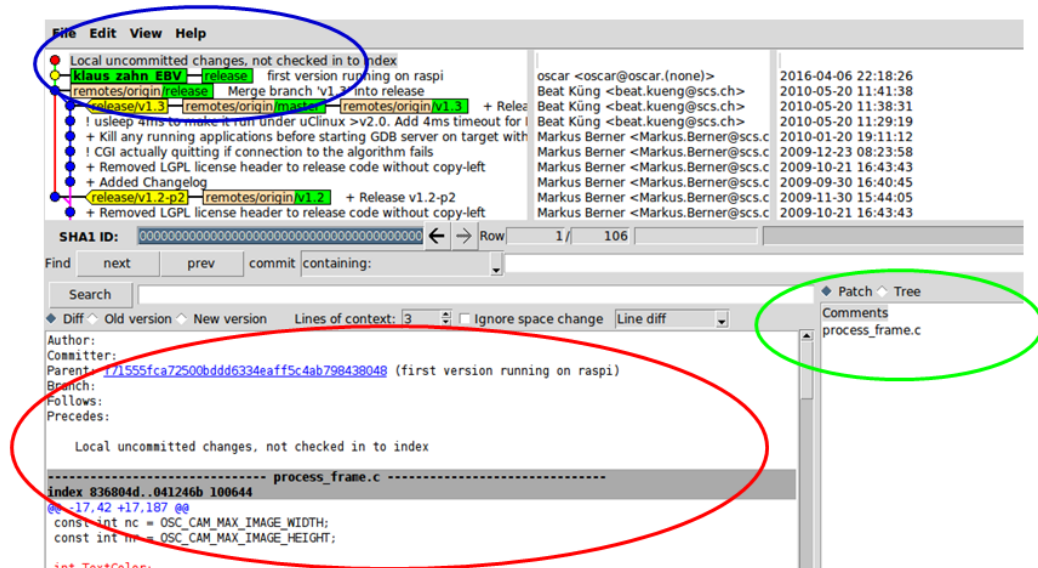
```
gitk --all
```

aus. Damit starten Sie eine graphische Benutzeroberfläche für die Visualisierung der Versionshistorie von `git`. Alle Angaben beziehen sich auf das *lokale* Repository `image_proc_agent.git`.

Das Fenster zeigt:



- Die verschiedenen Versionen (blau)
  - Wenn Sie eine der Versionen anwählen, die Liste der bei dieser Version geänderten Files, bezogen auf die Vorgängerversion (grün)
  - Wenn Sie eines der Files anwählen, die zugehörigen Änderungen im File (rot).
5. Sie sehen dann an dem roten Punkt (oberster Eintrag an der blau markierten Liste), dass es lokal geänderte Files gibt (welche -> grün) und können diese Änderungen nochmals kontrollieren (rot).

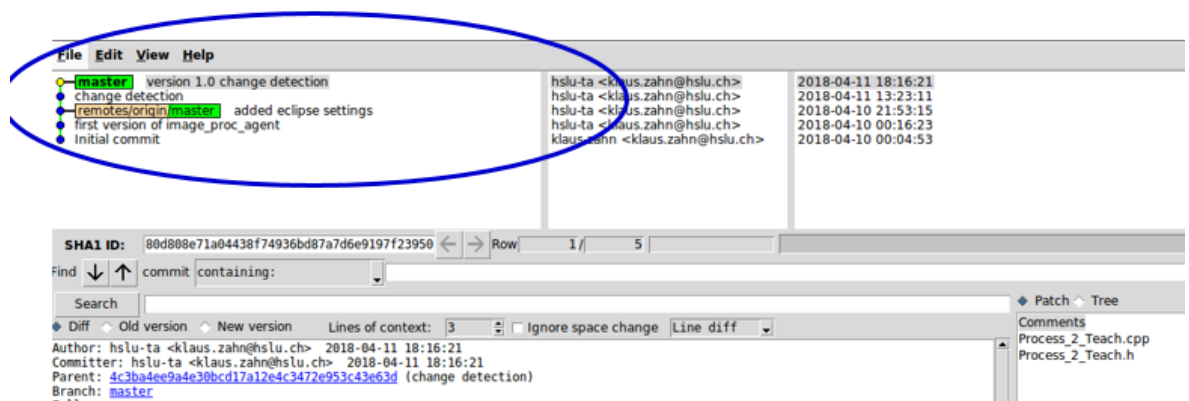


Für den commit geben Sie auf der Kommandozeile folgenden Befehl ein:

```
git commit -a -m 'version 1.0 change detection'
```

Hierbei können Sie die „commit Message“ hinter dem Argument `-m` nach Ihrem Geschmack anpassen.

6. Wenn Sie nun wieder mittels git die Versionshistorie untersuchen, so stellen Sie fest, dass ein neuer Eintrag hinzugekommen ist (blau):



Damit ist Ihre Arbeit (zumindest lokal) gesichert und Sie können bei weiteren Änderungen immer wieder darauf zurückgreifen.

7. Sie können die Anwendung nun erweitern, indem Sie die Hintergrundschätzung durch gleitenden Mittelwert aus Kapitel 2.4.2 [15] implementieren. Hierzu müssen Sie das gewichtete Mittel aus dem aktuellen Bild `inputImage` und der Hintergrundschätzung `mBkgrImage` (als Member Variable, um das Ergebnis zu speichern!) bilden. Dies können Sie wie folgt erreichen:

```
double alpha = 0.9;
cv::addWeighted(mBkgrImage, alpha, inputImage, 1 - alpha, 0, mBkgrImage);
```

Hierbei wird das Bild `mBkgrImage` mit dem Wert `alpha` multipliziert und das Bild `inputImage` mit dem Wert `1-alpha` und dann die Summe gebildet. Das Ergebnis wird nach `mBkgrImage` zurückgeschrieben. Mit Hilfe dieser Hintergrundschätzung können Sie dann einfach auf Basis der Change Detection die Matlab Implementierung aus Kapitel 2.4.2 nachbilden.

8. Falls Sie noch eine etwas anspruchsvollere Aufgabe suchen, so können Sie das «klassische» Region Labeling durch die modernere Variante basierend auf Kettencodes ersetzen (vgl. Sie hierzu auch Kapitel 5.2 in [15]). Hierzu dient die Funktion `cv::findContours( )`. Dabei können Sie das folgende Codesegment verwenden:

```
cv::findContours(binaryImage, contours, hierarchy, CV_RETR_EXTERNAL ,
                CV_CHAIN_APPROX_SIMPLE);

for(unsigned int idx = 0 ; idx < contours.size(); idx++ ) {
    //area
    double area = cv::contourArea(contours[idx]);
    //bounding rectangle
    cv::Rect rect = cv::boundingRect(contours[idx]);
    //center of gravity
    // center of mass
    cv::Moments moment = cv::moments(contours[idx]);
    double cx = moment.m10 / moment.m00;
    double cy = moment.m01 / moment.m00;

    //to draw counter to index idx in image
    cv::drawContours(resultImage, contours, idx, cv::Scalar(255), 1, 8 );
}
```

## 6.2. Übung 9: OCR

In dieser Übung werden wir die Anwendung zur Klassifikation/Kategorisierung am Beispiel OCR aus dem Kapitel 8.2 des Theorieskriptes [15] auf den Raspberry Pi portieren.

### Ziele:

1. Wir können komplexere Bildverarbeitungsalgorithmen auf dem Raspberry Pi implementieren.

2. Wir verstehen grundlegende Konzepte der Objektklassifikation.
3. Wir verstehen das Prinzip des HOG-Klassifikators («Histogram of Oriented Gradients», s. Kap. 8.2 in [15]).

### Übung im Detail:

1. Beim Studium der Matlab Files `OCR_Letter_train.m/`  
`OCR_Letter_classify.m` (Kap. 8.2 in [15]) ist erkenntlich, dass für die Erstellung des Feature Vektors folgende Schritte notwendig sind:

- a) die Bildung der Ableitungen in x- und y-Richtung mit einem Sobel Filter

```
ActImageDx = imfilter(double(Image), DX);
```

- b) die Berechnung des Canny Edge Filters

```
EdgeCanny = edge(Image, 'canny', Threshold, Sigma);
```

- c) das Binning der Kantenrichtungen nach den Winkeln

```
Angle = pi+atan2(ActImageDy, ActImageDx);  
Angle = 1+mod(round(Angle/(2*pi/AngleBins)), AngleBins);
```

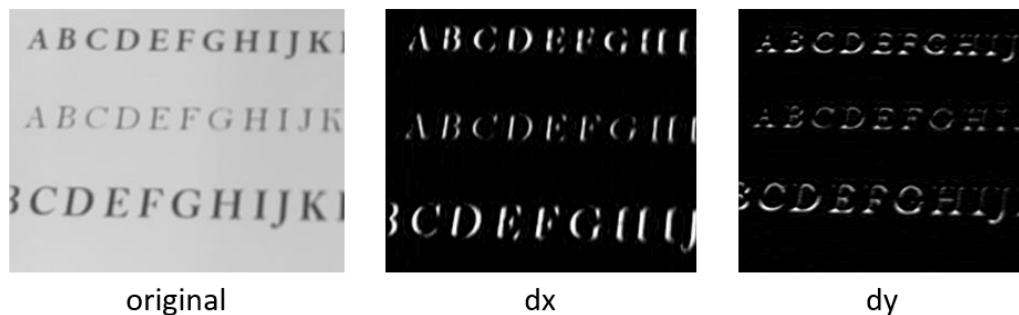
- d) das räumliche Binning der Kantenrichtungen  
(Loop über die Regions)

Wir wollen diese Punkte nun Schritt für Schritt erarbeiten.

2. *Ableitungen in x- und y-Richtung mit einem Sobel Filter*  
Die Ableitung mit dem Sobel Filter in OpenCV erfolgt mit den Befehlen:

```
cv::Sobel(inputImage, dxImage, CV_16S, 1, 0, 3);  
cv::Sobel(inputImage, dyImage, CV_16S, 0, 1, 3);
```

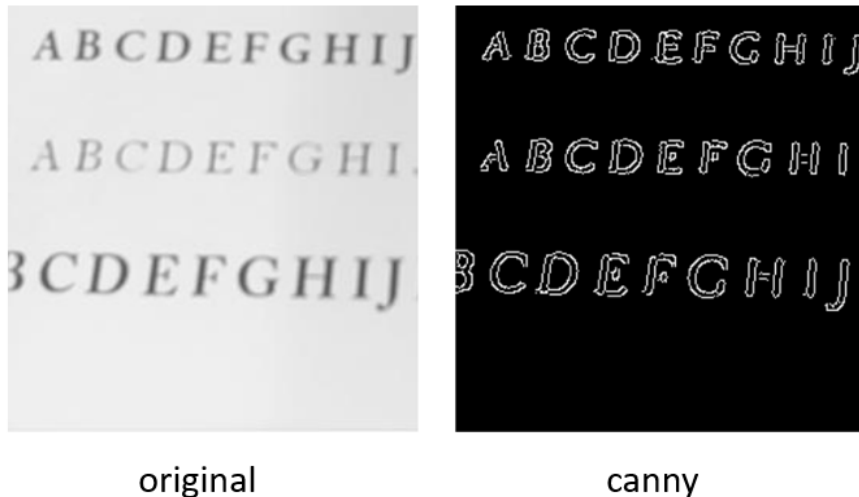
Dabei ist das Ausgabebild (`dxImage` bzw. `dyImage`) vom Type 16 bit sigend (`CV_16S`). Implementieren Sie diese beiden Ableitungen und visualisieren Sie das Resultat im VCA Web GUI. Das Ergebnis sollte etwa so aussehen:



3. *Berechnung des Canny Edge Filters*  
Für die Anwendung des Canny Filters wird das Bild vorab noch mit einem Tiefpassfilter geglättet:

```
cv::GaussianBlur(inputImage, blurImage, cv::Size(3, 3), 2);
cv::Canny(blurImage, cannyImage, 1, cannyThreshold, 3);
```

Bestimmen Sie manuell einen geeigneten Schwellwert für den Canny Algorithmus (`cannyThreshold`) mit Hilfe des Sliderwertes des VCA Web GUI (`threshold`) und setzen Sie diesen dann fix<sup>40</sup>. Visualisieren Sie hierzu das Resultat im GUI:



#### 4. *Binning der Kantenrichtungen nach den Winkeln*

Für die Darstellung des Ergebnisses werden acht verschiedene Farbwerte verwendet:

```
unsigned char colors[8][3] =
    { {255, 0, 0},{ 255, 191, 0},{ 128, 255, 0 },
      { 0, 255, 64 },{0, 255, 255 },{ 0, 64, 255 },
      { 128, 0, 255 },{ 255, 0, 191 } };
```

Zuerst wird das Ergebnisbild korrekt initialisiert (3 Byte Farbbild `CV_8UC3`):

```
cv::Mat angleCol(inputImage.rows, inputImage.cols, CV_8UC3);
angleCol.setTo(cv::Scalar(0, 0, 0));
```

Der Pixelloop sieht folgendermassen aus. Dabei führen wir die aufwendigen Operationen (v.a. `ArcTan`) nur für die Pixel mit positivem Canny Wert durch:

```
const int NumAngleBins = 8;

//loop over rows
for (int rows = 0; rows < inputImage.rows; rows++) {
    //loop over cols
    for (int cols = 0; cols < inputImage.cols; cols++) {
        ///do only work for canny results
        if (cannyImage.at<unsigned char>(rows, cols) > 0) {
            double angle = CV_PI + atan2(dyImage.at<short>(rows, cols),
                                          dxImage.at<short>(rows, cols));
```

<sup>40</sup> Den Slider für den Schwellwert benötigen wir dann für die Binarisierung des Bildes.

```

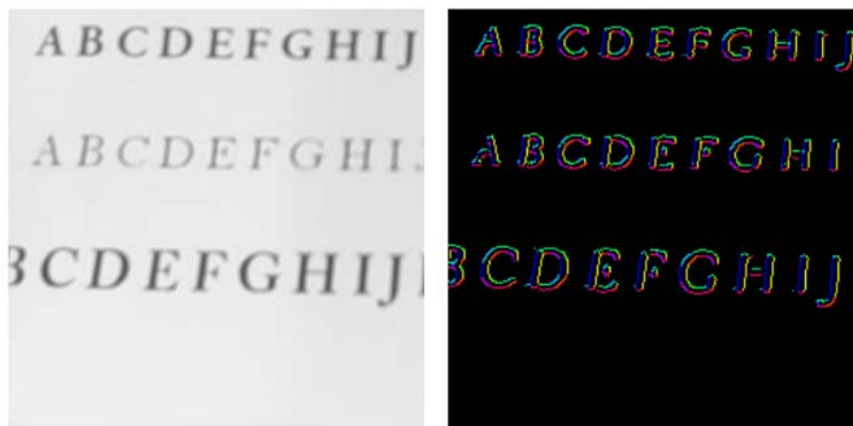
double angleD = 2.*CV_PI / NumAngleBins;
//angle binning
int dir = ((int)((angle / angleD) + 0.5)) % NumAngleBins;

cv::Vec3b col;
col[0] = colors[dir][0];
col[1] = colors[dir][1];
col[2] = colors[dir][2];

angleCol.at<cv::Vec3b>(rows, cols) = col;
    }
}
}

```

Visualisieren Sie das Ergebnis im Web GUI:



original

winkel binning

##### 5. *Räumliches Binning der Kantenrichtungen*

Hierzu führen wir eine Binarisierung des Eingabebildes durch, wobei Sie für den Schwellwert (wie in Übung 8) den Slider des Web GUI verwenden. Beachten Sie, dass Sie eine «inverse» Binarisierung durchführen müssen, da die Objekte dunkler sind als der Hintergrund.

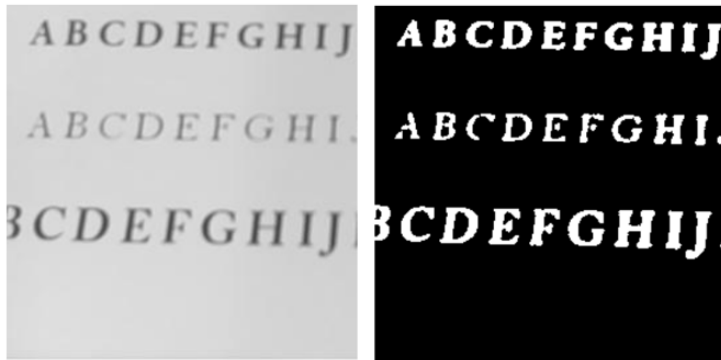
```

int threshold = this->mParams->Process2Teach_binaryThreshold.getValueAsInt();
cv::Mat binaryImage;
cv::threshold(inputImage, binaryImage, threshold, 255, CV_THRESH_BINARY_INV);

```

Visualisieren Sie das Ergebnis im Web GUI:





original

binarisierung

Wichtig hierbei ist zu beachten, dass das Folgende nur funktioniert, falls die Buchstaben klar segmentiert sind, d.h. insbesondere voneinander isoliert sind. Führen Sie nun wie in Übung 8 ein Region Labeling und die Merkmalsextraktion durch. Im Folgenden verwenden wir die Merkmale:

```
int topLeftx = stats.at<int>(i, 0);
int topLefty = stats.at<int>(i, 1);
int width = stats.at<int>(i, 2);
int height = stats.at<int>(i, 3);
//last entry is the area
int area = stats.at<int>(i, 4);
```

Hierbei ist *i* wieder der Index der Region (0 ist der Index des Hintergrundes)

Implementieren Sie dann einen Loop über die Regionen (d.h. *i*), wobei Sie nur «sinnvolle» Grössen innerhalb von zwei Grenzen zulassen.

```
if (minArea < area && area < maxArea) {
}
```

Bestimmen Sie dann für jede Region (= Buchstaben) den zugehörigen Feature Vektor<sup>41</sup>.

```
double FeatureVec[NumFeatures];
memset(FeatureVec, 0, sizeof(FeatureVec));
```

Hierzu ist das korrekte Binning durchzuführen, indem Sie über alle Pixel der Region iterieren. Wir verwenden der Einfachheit halber eine Iteration über die Bounding Box<sup>42</sup>.

```
for (int rows = topLefty; rows <= topLefty + height; rows++) {
    for (int cols = topLeftx; cols <= topLeftx + width; cols++) {
        //Do binning here
    }
}
```

<sup>41</sup> Fügen Sie an dieser Stelle das File `FeatureVectors.h`, welches vom Matlab Skript `OCR_Letter_train.m` erzeugt wird, inklusive `#include` Statement zum Projekt hinzu.

<sup>42</sup> Dies erfolgt analog der Matlab Implementierung und könnte ggf. bei nahe zusammen liegenden Buchstaben mit überlappender Bounding Box zu Problemen führen.

Das eigentliche Binning (an der gelb markierten Stelle einzufügen), ist wie folgt:

```
int bin = 0;
int rb = rows - topLeftty;           //0 1
int cb = cols - topLefttx;           //2 3    *8
if (width / 2 < cb) {                 //4 5
    bin += 1;
}
if ((2 * height) / 3 < rb) {
    bin += 4;
} else if (height / 3 < rb) {
    bin += 2;
}
bin = bin << 3; //Multiply by eight
FeatureVec[bin + dir] += 1.;
```

Hierbei ist `dir` der Wert des Winkel Bins (s. Punkt 4, oben).

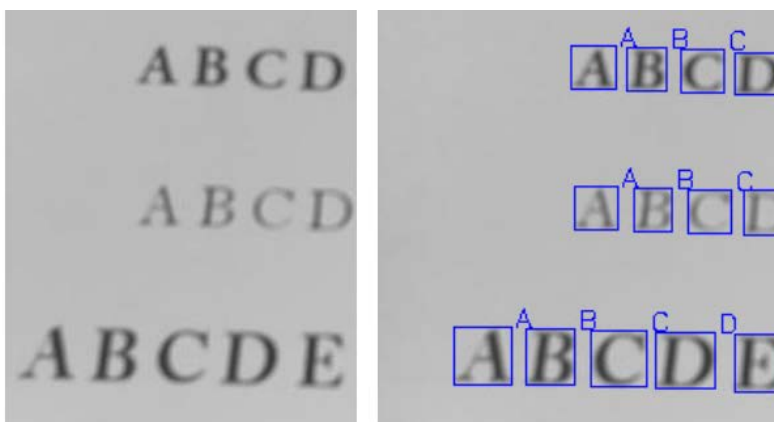
Nach dem Loop über die Bounding Box muss der Feature Vektor noch normiert werden:

```
double SumFeatVec = 0;
for (int i0 = 0; i0 < NumFeatures; i0++) {
    SumFeatVec += FeatureVec[i0];
}

for (int i0 = 0; i0 < NumFeatures; i0++) {
    FeatureVec[i0] = FeatureVec[i0] / SumFeatVec;
}
```

Nun kann durch Vergleich mit den mittleren Feature Vektoren aus dem Training im File `FeatureVectors.h` der beste Match (kürzeste Euklidische Distanz der Feature Vektoren) bestimmt werden.

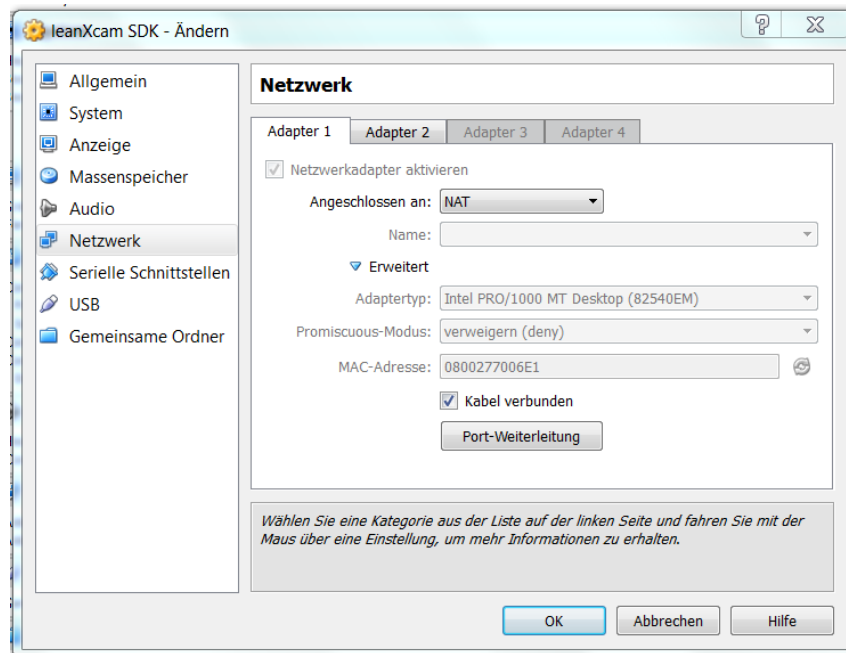
Stellen Sie abschliessend die Bounding Boxen und gefundenen Buchstaben noch im Web GUI dar:



## 7. Details zu VirtualBox

### 7.1. Netzwerk Einstellungen

Die Einstellungen werden unter dem Dialog *Ändern* (die zugehörige virtuelle Maschine muss ausgewählt sein) vorgenommen (s. Abbildung 20). Eine allgemeine Einführung zu den Netzwerkeinstellungen von VirtualBox findet sich in [16]. Ein grundsätzliches Verständnis der Wirkungsweise von NAT, Netzwerkbrücke und Host-only Adapter (unter *Angeschlossen an*) sollte vorhanden sein.



**Abbildung 20:** Dialog zur Einstellung der Netzwerk Settings.

Um sowohl eine Verbindung zum Internet (und zur Raspberry Pi, mit Default IP 192.168.1.10) als auch zum (und vor allem vom) Host zu ermöglichen, müssen zwei Netzwerkadapter mit unterschiedlichen Einstellungen konfiguriert werden:

1. Wir gehen davon aus, dass per Default der Netzwerkadapter 1 (Tab *Adapter 1* im Dialog) aktiviert und auf NAT eigestellt ist<sup>43</sup>.
2. Maschine ausschalten, um einen weiteren Netzwerkadapter (Tab *Adapter 2* im Dialog) zu aktivieren.
3. Checkbox *Netzwerkadapter aktivieren* auswählen und unter *Angeschlossen an* Host-only Adapter auswählen (Abbildung 21).
4. In der VM ggf. das Netzwerk neu starten: *service network-manager restart*
5. Falls es noch Probleme mit dem Internetzugriff gibt, die Routing Tabelle überprüfen: *netstat -r*

<sup>43</sup> Das Default Netzwerk für die NAT Verbindung ist 10.0.2.x/24 mit der Default Gast IP Adresse 10.0.2.15. Um diese Einstellungen zu ändern muss auf der Kommandozeile (des Host) der folgende Befehl eingegeben werden (Netzwerk natürlich anzupassen): *VBoxManage modifyvm "Name der VM" --natnet1 "192.168.55/24"*

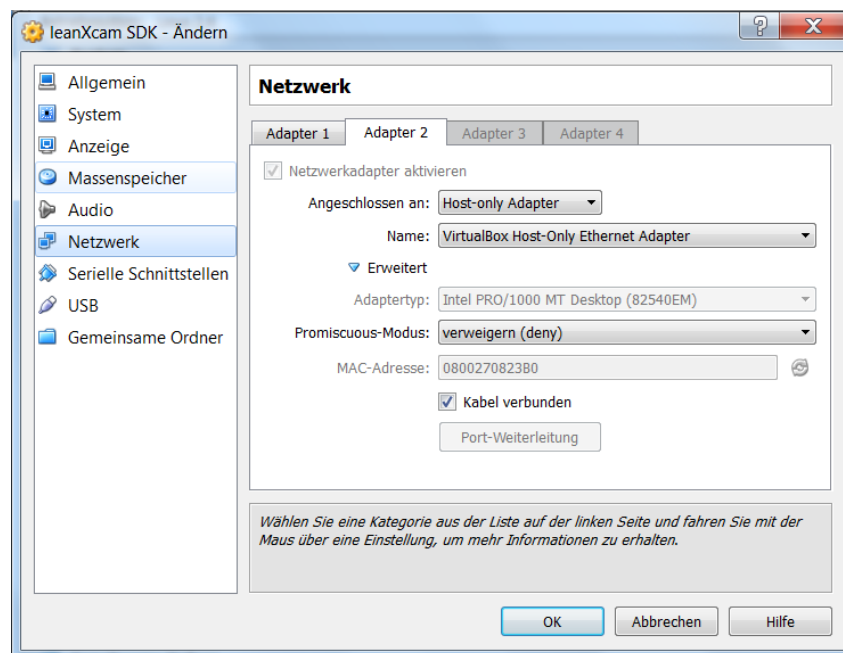
```

oscar@oscar-VirtualBox: ~
File Edit View Search Terminal Help
oscar@oscar-VirtualBox:~$ netstat -r
Kernel IP routing table
Destination      Gateway         Genmask         Flags         MSS Window  irtt Iface
default          10.0.2.2        0.0.0.0         UG            0 0        0 eth0
10.0.2.0         *               255.255.255.0   U            0 0        0 eth0
link-local       *               255.255.0.0     U            0 0        0 eth0
192.168.56.0     *               255.255.255.0   U            0 0        0 eth1
oscar@oscar-VirtualBox:~$

```

Die *default* Route muss zum NAT Gateway im 10.0.2.x/24 Netz zeigen.

6. Im „schlimmsten Fall“ die Route manuell eingeben:  
*sudo route add default gw 10.0.2.0 eth0*. Vorher ggf. die *default* Route mit  
*sudo route del default* löschen.



**Abbildung 21:** Einstellungen für Host-only Adapter.

## 8. Raspberry Pi

Hier finden sich noch Ergänzungen zur Verwendung von SDK, OSCar Framework und Raspberry Pi.

### 8.1. Starten der Default Applikation

Auf dem Raspberry Pi Kit ist die Applikation `vcagrabber` installiert, welche die Übermittlung von Bildern an den Webserver ermöglicht. Falls diese Applikation gestoppt wurde, kann Sie mit

```
sudo /etc/init.d/vcagrabber-init start
```

wieder aktiviert werden.

### 8.2. Verbindung ohne Passwort Eingabe

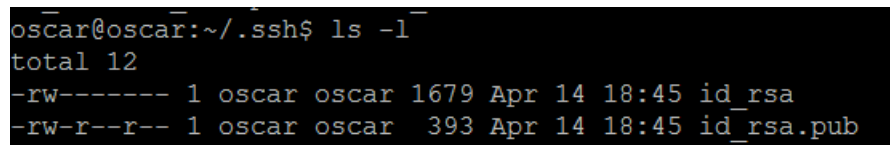
Die Eingabe des Passworts für die ssh-Verbindung zum Raspberry Pi ist auf Dauer etwas mühsam. Durch Austausch eines Schlüssels kann dies umgangen werden. Hierzu folgende Schritte durchführen:

1. Wechseln Sie auf der Ubuntu VM in das Verzeichnis `/home/oscar/.ssh`:<sup>44</sup>  
`cd ~/.ssh`

2. Führen Sie den folgenden Befehl aus und quittieren Sie alle Meldungen mit Enter:

```
ssh-keygen
```

3. Hiermit erzeugen Sie im aktuellen Verzeichnis ein Schlüsselpaar. Verifizieren Sie mittels `ls` die Existenz der beiden Files `id_rsa` und `id_rsa.pub`:



```
oscar@oscar:~/.ssh$ ls -l
total 12
-rw----- 1 oscar oscar 1679 Apr 14 18:45 id_rsa
-rw-r--r-- 1 oscar oscar  393 Apr 14 18:45 id_rsa.pub
```

Der private Schlüssel liegt im File `id_rsa` und `id_rsa.pub` enthält den public Schlüssel („private key“ und „public key“).

4. Den public key kopieren Sie nun mittels `scp` auf den Raspberry Pi:

```
scp id_rsa.pub pi@192.168.1.10:/home/pi/.ssh
```

5. Verbinden Sie sich nun per `ssh` (hoffentlich zum letzten Mal mit Passwort Eingabe) mit dem Raspberry pi und wechseln Sie in das Verzeichnis `/home/pi/.ssh`:

```
ssh pi@192.168.1.10
```

---

<sup>44</sup> Verzeichnisse welche mit einem Punkt beginnen werden mit `ls` nicht angezeigt. Hierzu müssen Sie `ls -la` eingeben.

```
cd .ssh
```

6. Nun führen Sie folgenden Befehl aus:

```
cat id_rsa.pub >> authorized_keys
```

Hiermit kopieren Sie den public key in das File `authorized_keys`.

7. Wenn nun eine `ssh`-Verbindung von der Ubuntu VM aufgebaut wird, so passt der private key der VM zum public key auf dem Raspberry Pi und die Verbindung wird ohne Passwort Eingabe initiiert. Testen Sie es.

## 9. Annex

### 9.1. References

- [1] [https://de.wikipedia.org/wiki/Raspberry\\_Pi](https://de.wikipedia.org/wiki/Raspberry_Pi)
- [2] <https://de.wikipedia.org/wiki/ARM-Architektur>
- [3] [http://de.wikipedia.org/wiki/Classless\\_Inter-Domain\\_Routing](http://de.wikipedia.org/wiki/Classless_Inter-Domain_Routing)
- [4] <http://www.boa.org>
- [5] <https://www.virtualbox.org/wiki/Downloads>
- [6] <https://www.raspberrypi.org>
- [7] <http://opencv.org>
- [8] <https://docs.opencv.org/3.0-beta/genindex.html>
- [9] <https://www.scs.ch/ueber-scs/departments/johannes-gassner/leanxcam.html>
- [10] <http://git-scm.com/doc>
- [11] <http://www.stack.nl/~dimitri/doxygen>
- [12] <https://github.com>
- [13] <http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Standards.html>
- [14] <http://www.ijon.de/comp/tutorials/makefile.html>
- [15] TA.BA\_EBV\_Skript.pdf
- [16] <http://www.dedoimedo.com/computers/virtualbox-network-sharing.html>
- [17] [https://github.com/scs/Raspberry\\_Pi/wiki/Users-Guide-Board-connectivity](https://github.com/scs/Raspberry_Pi/wiki/Users-Guide-Board-connectivity)
- [18] <http://de.wikipedia.org/wiki/Public-Key-Authentifizierung>

### 9.2. Abkürzungen

Liste der verwendeten Abkürzungen:

SDK	Software Development Kit
DSP	Digitaler Signalprozessor
VM	Virtuelle Maschine
SSH	Secure Shell
IPC	Interprozess Kommunikation
URL	Uniform Resource Locator
IDE	Integrated Development Environment
NAT	Network Address Translation
FPU	Floating Point Unit

### 9.3. git in a Nutshell

Für eine sehr gute Einführung in `git` sehen Sie unter [10] nach.  
Einige wichtige `git` Befehle seien hier in Kürze zitiert:



1. `git clone git://github.com/scs/app-template.git:`  
Erstelle eine Kopie des angegebenen (Remote) Repositories<sup>45</sup>.
2. `git clone /home/oscar/leanXcamSDK/app-template/.git:`  
Erstelle eine Kopie des angegebenen (lokalen) Repositories.
3. `gitk --all:`  
Starte die graphische Benutzeroberfläche zur Darstellung der Versionshistorie.
4. `git commit -a -m „Commit Message“:`  
Übertrage die Änderungen in das Repository unter der Meldung Commit Message.
5. `git add .:`  
Füge alle aktuell nicht versionierten („untracked“) Files zum „Staging Area“, um sie beim nächsten „Commit“ in Repository zu übertragen.
6. `git log:`  
Kommandozeilenausgabe der Versionshistorie.
7. `git checkout release:`  
Checke den Branch/Tag `release` aus.
8. `git tag -a v1.4 -m 'Message':`  
Erzeuge den Tag `v1.4` („Annotated Tag“) mit der Nachricht `'Message'`.
9. `git checkout -b BranchName:`  
Erzeuge einen neuen Branch namens `BranchName` und checke diesen aus (d.h. ein Commit erfolgt in den Branch).
10. `git merge Branchname:`  
Merge den Branch namens `BranchName` in den Branch `release`<sup>46</sup>.
11. `git branch:`  
Zeige aktuelle Branches an, wobei der aktuell ausgecheckte Branch mit einem Stern gekennzeichnet ist.
12. `git push --all git@github.com:user/rep.git:`  
Alle Branches (in `.git/refs/heads`) werden auf das angegebene (github<sup>47</sup> Remote) Repository übertragen (sog. „Push“). Alternativ zur Option `--all` kann auch ein spezieller Branch angegeben werden.
13. `git push --tags git@github.com:user/rep.git:`  
Alle Tags (in `.git/refs/tags`) werden auf das angegebene (github<sup>47</sup> Remote) Repository übertragen (sog. „Push“). Alternativ zur Option `--tags` kann auch ein spezieller Tag angegeben werden.
14. `git pull git://github.com/user/rep.git master` oder  
`git pull origin master:`

---

<sup>45</sup> Falls die Meldung „warning: remote HEAD refers to nonexistent ref, unable to checkout.“ erfolgt, so genügt es, einen Checkout auf eine neuen – selbst definierten – Branch durchzuführen.

<sup>46</sup> Wir nehmen an, dass wir aktuell auf dem Branch `release` arbeiten.

<sup>47</sup> Die Syntax der URL variiert je nach Server.

Der Branch `master` wird vom angegebenen (`github`<sup>47</sup> Remote) Repository (oder vom „origin“, d.h. vom Repository, welches mit „clone“ kopiert wurde<sup>48</sup>) heruntergeladen und in den aktuellen Branch eingefügt (merge).

15. `git pull -t git://github.com/user/rep.git master` oder  
`git pull -t origin master:`

Zusätzlich zu dem Branch `master` werden vom angegebenen (`github`<sup>47</sup> Remote) Repository alle verfügbaren Tags heruntergeladen.

16. `git fetch origin master`  
`git merge sha1`

Der Branch `master` wird vom angegebenen (Remote) Repository heruntergeladen und die Version korrespondierend zum angegebenen Hash `sha1` wird in den aktuellen Branch eingefügt (merge). Allerdings führt dies zu einem „detached HEAD“<sup>49</sup>.

17. `git reset --hard rev:`

Setze den Zustand des Repositories zurück auf `rev` und lösche alle Änderungen, welche seither durchgeführt wurden. Für `rev = HEAD` können Änderungen/Konflikte nach einem missglückten pull-Versuch rückgängig gemacht werden.

18. `git revlog:`

(und Verwalte) die Historie der Änderungen am Repository.

19. `ssh -vT git@github.com:`

Teste Verbindung zum github auf korrekte Verwendung des ssh-Keys.

20. `git remote add origin git://github.com/user/rep.git:`

Setzt das Remote Repository auf die angegebene URL. D.h. alle `git push/pull origin`

Befehle erfolgen dann bez. dieses Remote Repositories. Der aktuell wird wir so angezeigt:

`git remote -v`

### 9.3.1. Verwendung von github

Im vorangehenden Kapitel wurden verschiedene Methoden vorgestellt, um ein lokales Repository in ein Remote Repository zu übertragen. Die Erstellung eines Remote Repositories unter `github`<sup>50</sup> ist dabei eine interessante Option, da ein öffentlich zugängliches Repository kostenlos ist. Dabei ist allerdings für das Hochladen („pushen“) der Daten eine Authentifizierung notwendig, welche durch ein Paar von private und public Key erfolgt. Dabei wird der private Key lokal im Verzeichnis `~/.ssh` abgelegt, während der public Key auf `github` geladen werden muss. Das Vorgehen ist dabei wie folgt:

1. Erstellen eines private und public Key Paares<sup>51</sup>:

Wechseln Sie in das Verzeichnis `~/.ssh` und geben Sie auf der Kommandozeile folgenden Befehl ein:

<sup>48</sup> Die Definition von „origin“ findet sich in `.git/config`.

<sup>49</sup> Der „HEAD“ (`cat .git/HEAD`) zeigt immer auf den aktuellen Commit. Ein „detached HEAD“ bedeutet, dass der HEAD nicht zu einem Branch korrespondiert. Dies ist dann ein Problem, wenn Änderungen ins Repository übertragen werden sollen (commit), da diese dann potentiell (nach einem Ändern des „HEAD“) verloren gehen können.

<sup>50</sup> <https://github.com/>

<sup>51</sup> Falls ein Key Paar schon existiert, kann dieser Schritt übersprungen und direkt zu Punkt 2. übergegangen werden.

```
ssh-keygen -C email
```

Wobei email mit der Email Adresse zu ersetzen ist, welche für die Erstellung des Git Accounts unter github verwendet wurde. Es erscheint die Frage nach dem File, in welches der Key gespeichert werden soll;

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/oscar/.ssh/id_rsa):
/home/oscar/.ssh/id_rsa already exists.
Overwrite (y/n)? y
```

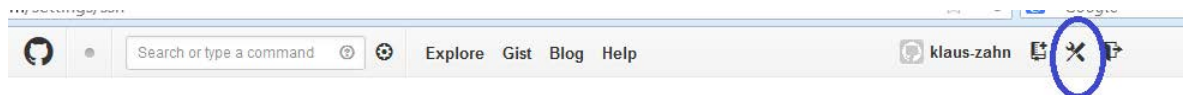
Beantworten Sie diese mit Enter, wobei Sie ggf. ein existierendes File überschreiben müssen. Hierauf antworten Sie ebenfalls mit „y(es)“.

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/oscar/.ssh/id_rsa.
Your public key has been saved in /home/oscar/.ssh/id_rsa.pub.
The key fingerprint is:
dc:4c:b3:d0:7f:9b:60:f8:87:cc:ce:dc:fc:82:be:72 email
```

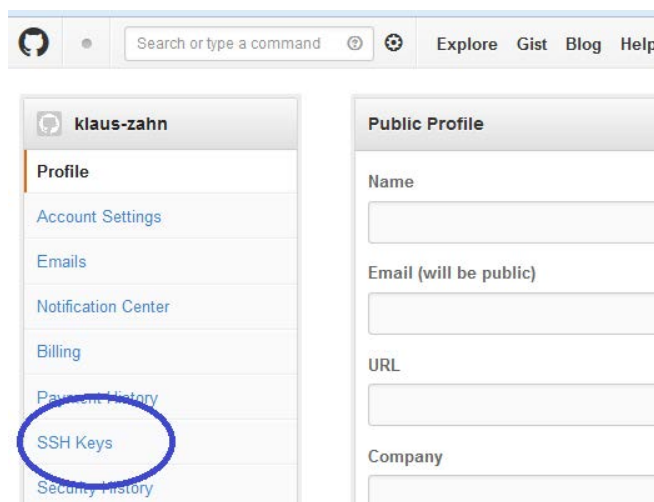
Danach erscheint die Frage nach einem Passwort, um das private Key File zu schützen. Hier sollten Sie aus Sicherheitsgründen ein Passwort eingeben, damit bei einem unerlaubten Zugriff auf Ihren Computer das private Key File geschützt ist. Sie müssen das Passwort nochmals wiederholen. Damit sind die Key Files (id\_rsa und id\_rsa.pub) erzeugt.

## 2. Registrieren des private Key in github:

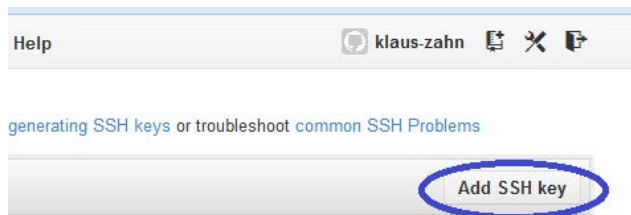
Für eine authentifizierte Verbindung mit github (Voraussetzung für einen Schreibzugriff) muss der private Key auf github hinterlegt werden. Hierzu in github unter Account settings (blaue Markierung)



die Option SSH Keys auswählen:



Dann die Option Add SSH key wählen



und den Inhalt von `id_rsa.pub` (eben den public Key) in das dafür vorgesehene Feld kopieren.

Nach dem Erstellen eines Repositories können dann mittels der im obigen Abschnitt 9.3 unter den Punkten 12 und 13 angegebenen Befehlen die Inhalte des lokalen Repositories in das Remote Repository übertragen werden.

## 9.4. Trouble Shooting

### 9.4.1. Manual Page

Um unter Linux die Manual Page z.B. zum Befehl `ssh` aufzurufen, tippen Sie:

```
man ssh
```

Sie können dann auf der Manual Page mit den Pfeiltasten  $\uparrow\downarrow$  navigieren, sowie mittels Eingabe von `Slash`

/

nach einem Pattern suchen (mit jedem Drücken der `n`-Taste springen Sie zum nächsten Match).

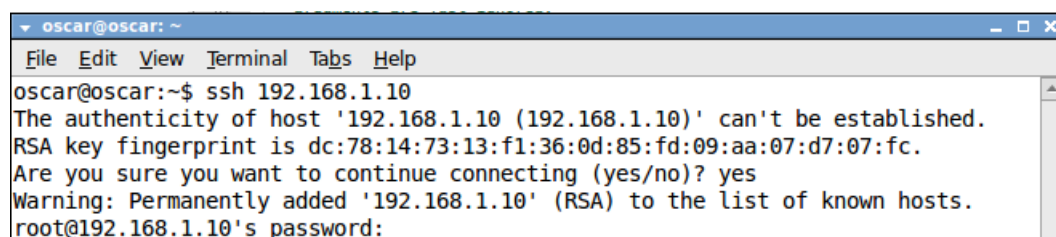
Die Manual Page verlassen Sie mit der `q`-Taste.

### 9.4.2. Issues bei der SSH-Verbindung mit dem Raspberry Pi

#### Meldung:

```
The authenticity of host '192.168.1.10 (192.168.1.10)' can't be established.
RSA key fingerprint is dc:78:14:73:13:f1:36:0d:85:fd:09:aa:07:d7:07:fc.
Are you sure you want to continue connecting (yes/no)?
```

SSH nutzt die Public Key Authentisierung [18], um die Identität des Computers, zu dem die Verbindung hergestellt wird, zu verifizieren. Bei der ersten Verbindung erscheint die obige Meldung.



Diese mit `yes` bestätigen<sup>52</sup>. Dann wird der Public Key des Computers im File

`~/.ssh/known_hosts`

gespeichert. Daher wird bei einem späteren Verbindungsaufbau die obige Meldung nicht mehr erfolgen, da die Authentizität auf Basis des gespeicherten Key verifiziert werden kann.

### **Meldung:**

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@      WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!      @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
95:69:1e:d8:55:fb:09:4d:54:c2:7c:cb:bc:0d:56:12.
Please contact your system administrator.
Add correct host key in /home/oscar/.ssh/known_hosts to get rid of this
message.
Offending ECDSA key in /home/oscar/.ssh/known_hosts:1
  remove with: ssh-keygen -f "/home/oscar/.ssh/known_hosts" -R 192.168.1.10
ECDSA host key for 192.168.1.10 has changed and you have requested strict
checking.
Host key verification failed.
lost connection
```

Der Key im File `~/.ssh/known_hosts`, der bei ersten Verbindung mit dem Computer 192.168.1.10 gespeichert wurde, stimmt nicht mit dem aktuell vom Computers 192.168.1.10 übermittelten Key überein. Dies könnte potentiell bedeuten, dass jemand versucht, Sie als Computer 192.168.1.10 auszugeben.

In unserem Fall heisst dies lediglich, dass die VM mit einer anderen Raspberry Pi HW erstellt wurde. Um das Problem zu lösen, einfach den entsprechenden Eintrag im File `known_hosts` löschen:

```
ssh-keygen -f "/home/oscar/.ssh/known_hosts" -R 192.168.1.10
```

### **9.4.3. Raspberry Pi Kamera sendet keine Bilder**

Falls der Raspberry Pi prinzipiell ansprechbar ist (z.B. über ssh) aber keine Bilder geliefert werden (vor allem auch nach dem Reboot, wenn eigentlich die Default Applikation laufen sollte, s. Abbildung 7), dann ist vermutlich der Stecker auf dem Kamera PCB lose (Abbildung 22).

Um das Problem zu lösen, den Raspberry Pi von der Stromversorgung trennen, das Gehäuse der Kamera öffnen und den Stecker vorsichtig in die Buchse drücken. Dann bei geöffnetem Gehäuse<sup>53</sup> den Raspberry Pi wieder starten und die Funktionalität verifizieren. Schliesslich das Kameragehäuse wieder schliessen.

<sup>52</sup> Damit wird die Authentizität des Ziel Computers manuell bestätigt.

<sup>53</sup> Beim Schliessen des Kameragehäuses rutscht der Stecker leicht heraus.



Abbildung 22: Stecker der Raspberry Pi Kamera.

#### 9.4.4. Gemeinsamer Ordner nicht verfügbar

Zuerst einen Reboot der Ubutnu VM (Maschine -> Zurücksetzen) durchführen. Dann mittels des Befehls `mount` die aktuell verbundenen Laufwerke anzeigen. Der „Gemeinsame Ordner“ sollte angezeigt sein (im Bsp. untern wurde der Name `share` im Filedialog in Abbildung 14 verwendet).

```
oscar@oscar:/media/share$ mount
/dev/sda1 on / type ext3 (rw,relatime,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
/sys on /sys type sysfs (rw,noexec,nosuid,nodev)
varrun on /var/run type tmpfs (rw,noexec,nosuid,nodev,mode=0755)
varlock on /var/lock type tmpfs (rw,noexec,nosuid,nodev,mode=1777)
udev on /dev type tmpfs (rw,mode=0755)
devshm on /dev/shm type tmpfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
lrn on /lib/modules/2.6.24-24-generic/volatile type tmpfs (rw)
securityfs on /sys/kernel/security type securityfs (rw)
share on /media/share type vboxsf (rw)
gvfs-fuse-daemon on /home/oscar/.gvfs type fuse.gvfs-fuse-daemon (rw,nosuid,nodev,user=oscar)
```

Falls der gewünschte Ordner nicht aufgeführt ist, kann versucht werden, den „Gemeinsamen Ordner“ manuell zu mounten:

```
sudo mount -t vboxsf share /media/share
```

Hier wird der „Gemeinsame Ordner“ `share` im Verzeichnis `/media` ge-mounted, d.h. die enthaltenen Files könnten dann mittels `ls -l /media/share` angezeigt werden.

Falls dies nicht funktioniert, die Konfiguration von Abbildung 14 nochmals durchführen. Ggf. muss ein neues Verzeichnis auf dem Host gewählt werden.

#### 9.4.5. Keine Schreibrechte auf dem Gemeinsamen Ordner

Dieses Problem trat bei Mac OS Betriebssystemen auf.

Den Gemeinsamen Ordner mit Lese- und Schreibrechten mounten (ggf. vorher unmounten mittels `umount share`):

```
sudo mount -t vboxsf -o rw,uid=1000,gid=1000 shared /media/share
```



### 9.4.6. Die Alt Taste funktioniert in der VM nicht

Lösung:

1. Oracle VM VirtualBox Manager öffnen->Globale Einstellungen -> Eingabe
2. Host-Taste wechseln zu Linker Windows Taste
3. OK Drücken

### 9.4.7. USB Stick mit der VM verbinden

Um einen USB Stick mit der VM zu verbinden, sollten das „VirtualBox Extension Pack“ installiert sein. Dies kann unter [5] heruntergeladen werden. Die Installation erfolgt durch eine Doppelklick auf das File.

Nach der Installation des Extension Pack die Ubuntu VM herunterfahren und gemäss Abbildung 23 die USB 2.0 Einstellung für die VM aktivieren. Dann die VM wieder starten.

Nach dem Start kann ein Memory Stick – oder allgemeiner jedes USB-Device – wie in Abbildung 24 gezeigt verbunden werden. Ein Memory Stick erscheint dann als „Media“ auf dem Desktop und sollte unter `/media` gemounted sein.

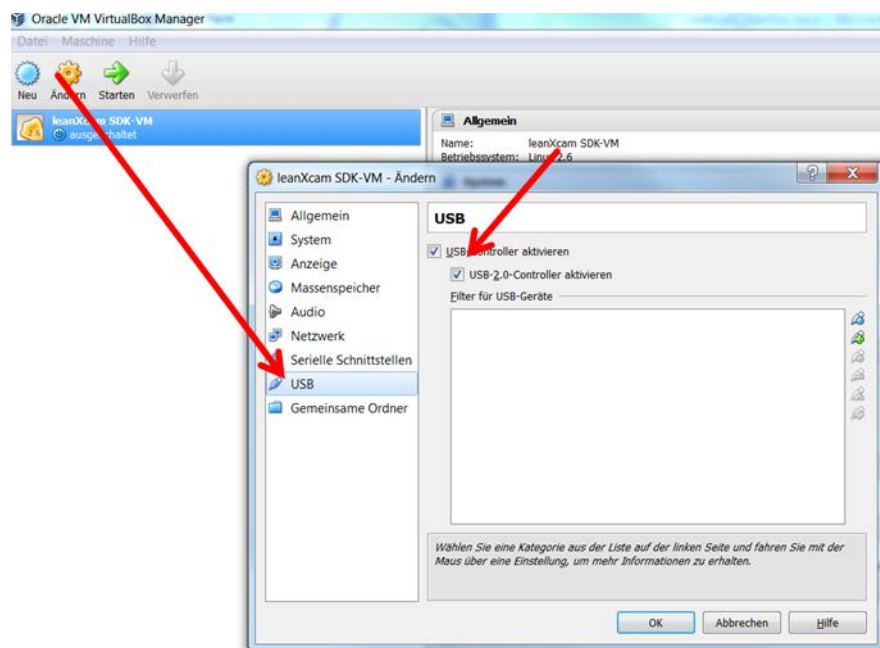
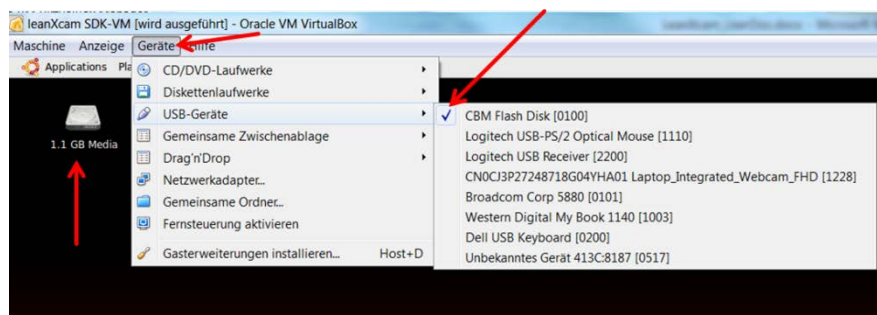


Abbildung 23: Setzen der USB Einstellungen der VM.

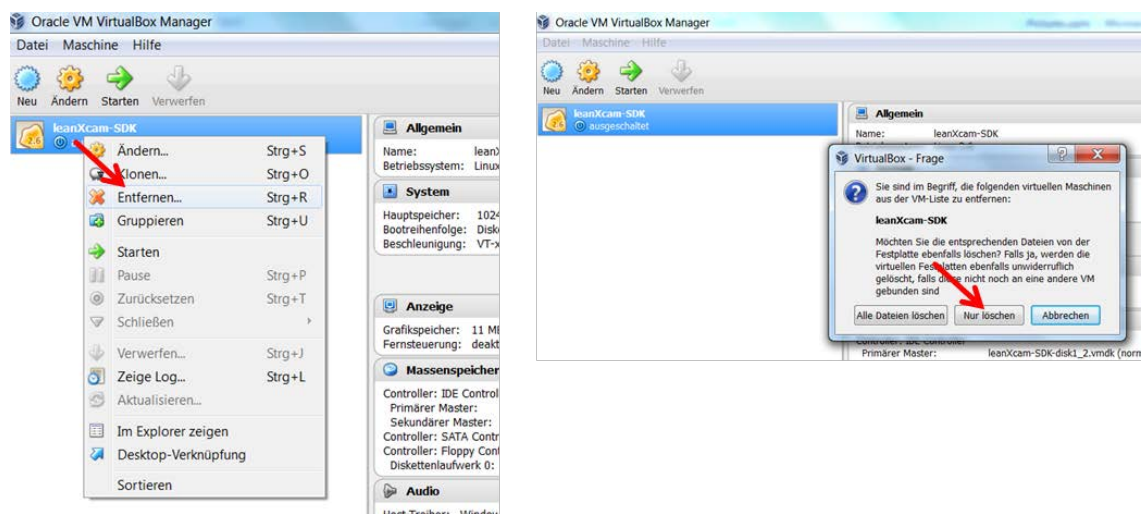




**Abbildung 24:** Verbinden eines UBS Devices (hier ein Memory Stick).

### 9.4.8. Falsche Namen der Netzwerk Adapter der VM (nicht `eth0` und `eth1`)

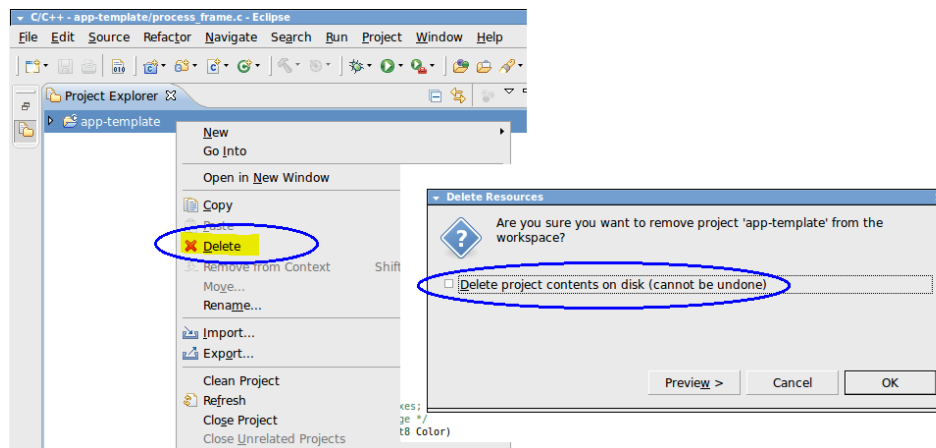
Falls nach dem Import der VM in VirtualBox die Netzwerk Schnittstellen nicht auf `eth0` bzw. `eth1` gesetzt sind (via `ifconfig` auf einem Terminal in der VM testen) so liegt ggf. ein Konflikt mit einer anderen VM vor. Dann bereits existierende VMs aus der VirtualBox entfernen (Abbildung 25, Bemerkung unter der Abbildung beachten), die Ubuntu VM löschen (d.h. gemäss Abbildung 25 vorgehen, aber Alle Dateien entfernen auswählen) und dann die Raspberry Pi-SDK VM erneut gemäss Kapitel 3 importieren.



**Abbildung 25:** Entfernen einer VM. **Vorsicht:** Alle Dateien entfernen löscht auch die Files auf der Festplatte.

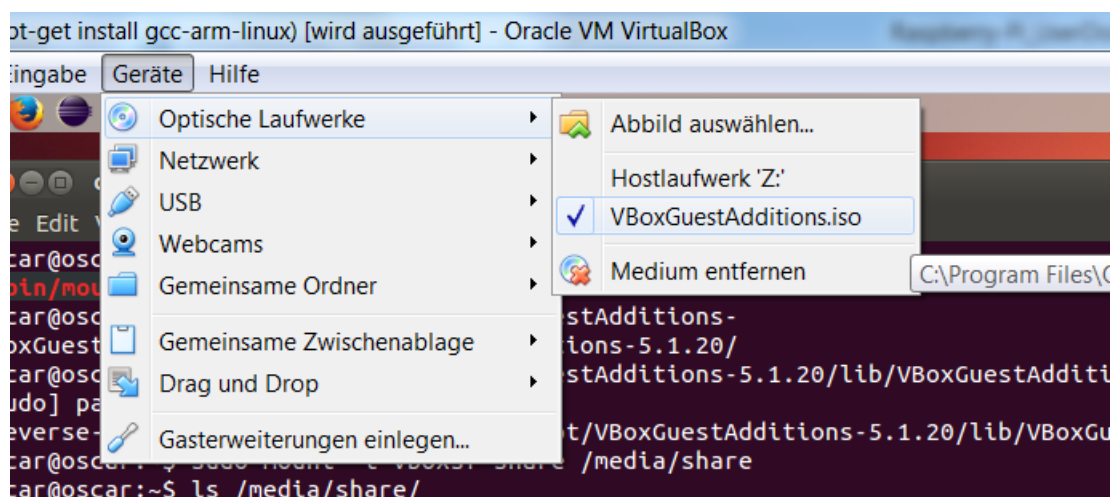
### 9.4.9. Konflikt unter Eclipse IDE beim Öffnen eines Projekts

Es kann vorkommen, dass nach dem Klonen eines Remote Repositories, welches den gleichen Namen verwendet wie ein zuvor unter Eclipse geöffnetes lokales Repository ein Konflikt beim Öffnen auftritt. Dies liegt an Projektinformationen, die von Eclipse im Verzeichnis `~/workspace` angelegt werden. Falls ein solcher Konflikt auftritt, so kann er durch Löschen des ursprünglichen (lokalen) Projektes behoben werden:



### 9.4.10. Fehler beim Mounten des Gemeinsamen Ordners

Nach einem Update der „Guest Additions“



kann es beim Versuch, den Gemeinsamen Ordner zu mounten zu folgender Fehlermeldung kommen:

```
mount: wrong fs type, bad option, bad superblock on /home/richard/Xdata,
missing codepage or helper program, or other error
In some cases useful info is found in syslog - try
dmesg | tail or so
```

Grund dafür ist, dass der symbolische Link von

`/sbin/mount.vboxsf`

„gebrochen“ ist.

Dieser kann mit folgendem Befehl wieder erstellt werden (beim gelb markierten Bereich die korrekte Version wählen):

```
sudo ln -f -s /opt/VBoxGuestAdditions-
***/lib/VBoxGuestAdditions/mount.vboxsf /sbin/mount.vboxsf
```

## 9.5. Erstellung der VM “from scratch”

Ubuntu 16.04 LTS herunterladen von:

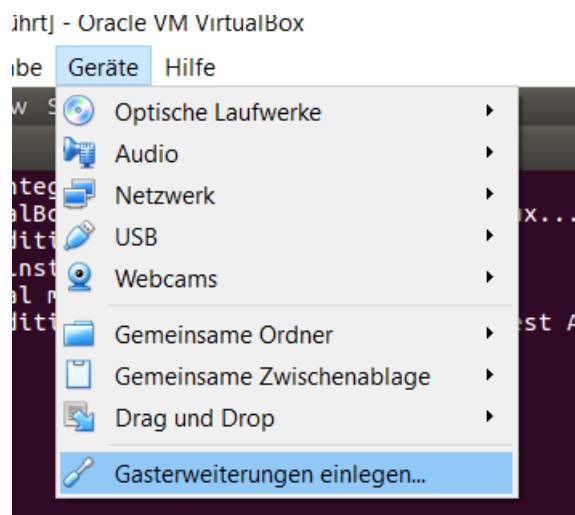
<https://www.ubuntu.com/download/desktop/contribute?version=16.04.3&architecture=amd64>

user: pi

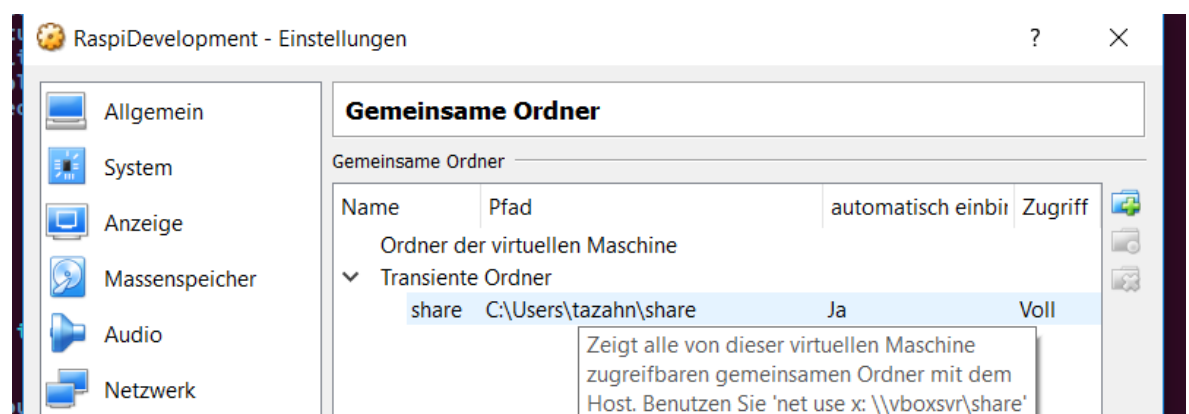
pwd: hslu (not required for login)

Anforderungen gemäss:

- 2 GHz dual core processor or better
- 1 GB system memory
- 10 GB of free hard drive space
- guest additions hinzufügen:



- share drive in Konfiguration hinzufügen:



- Verzeichnis erstellen:  
`sudo mkdir /media/share`

- **Share drive mounten:**  
`sudo mount -t vboxsf shared /media/share`
- **Update:**  
`sudo apt-get update`  
`sudo apt-get upgrade`
- **git installieren:**  
`sudo apt-get install git`
- **gitk installieren:**  
`sudo apt-get install gitk`
- **java jre installieren:**  
`sudo apt-get install default-jre`
- **Eclipse herunterladen von:**  
<https://www.eclipse.org/downloads/>
- **Eclipse installieren:**  
Installer entpacken und starten: `./eclipse-inst`  
C/C++ Package und dann weiter ...
- **Eclipse Icon kopieren:**  
`sudo cp /home/hslu-ta/eclipse/cpp-oxygen/eclipse/icon.xpm`  
`/usr/share/pixmaps/eclipse.xpm`  
und Eclipse an Launcher "pinnen"
- **ssh-server installieren:**  
`sudo apt-get install openssh-server`
- **sshpass installieren:**  
`sudo apt-get install sshpass`
- **Cross Compiler installieren:**  
`apt-get install gcc-arm-linux-gnueabi`  
`apt-get install g++-arm-linux-gnueabi`
- **Applikation Template image\_proc\_agent installieren**  
checkout Repository:  
`git clone https://github.com/klaus-zahn/image\_proc\_agent.git`  
  
**test build x86**  
`make`  
  
**test build arm**  
`TARGET_TYPE=arm make`