Hochschule Luzern
Technik & Architektur

Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur
CC Innovation in Intelligent
Multimedia Sensor Networks

Benutzer Dokumentation für leanXcam

Document Status: on-goging
Issue Date: 24.04.2014
Author(s): Klaus Zahn

Document History

Revision	Date	Changes	Author
0.9	12.03.2013	Document Setup	Klaus Zahn
0.91	20.03.2014	Übung 8 hinzugefügt	Klaus Zahn
0.92	23.04.2014	Übung 9 hinzugefügt	Klaus Zahn

Contents

1. Einleitung

Die leanXcam Plattform könnte kurz als „programmierbare IP-Kamera“ bezeichnet werden. Die grundsätzliche Idee hinter der Entwicklung der leanXcam und deren wesentliche Charakteristiken sind die Folgenden [1]:

Abbildung 1: *Die leanXcam HW.*

- Bereitstellung einer „intelligenten“ d.h. programmierbaren Kamera für Visionsanwendungen (d.h. Bildverarbeitung im industriellen Umfeld) zu einem vergleichsweise niedrigen Stückpreis.

- Die Kamera kann als vollständig autonomes System funktionieren und zum Beispiel zur Qualitätskontrolle direkt in eine Maschine eingebaut werden. Alle Berechnungen werden dabei direkt auf der Kamera durchgeführt (eben „intelligente Kamera“).

- Das zur leanXcam gehörige Programmierframework sowie die Hardware-Pläne der Kamera werden Open-Source zur Verfügung gestellt. Es existiert eine kostenlosen Online-Plattform [2] zum Austausch mit anderen leanXcam-Entwicklern und zum Download von Updates sowie Beispielapplikationen.

Auf der Online-Plattform [2] finden sich zahlreiche Dokumente zur Programmierung und Verwendung der leanXcam sowie die SW-Repositories mit allen Quelldateien [3]. Trotzdem sollen hier die wesentlichen Schritte – mit verschiedenen spezifischen Ergänzungen (in Bezug auf [2]) – zusammengefasst werden, um ein konsistentes Dokument für die Einarbeitung in Plattform und deren Verwendung zur Verfügung zu stellen.

2. Inbetriebnahme

2.1. Das leanXcam Kit und Zusatz SW

Das leanXcam Kit besteht aus:

1. leanXcam HW
2. Netzadapter (kann auch durch das in der Abb. nicht gezeigte USB-Kabel ersetzt werden)
3. Ethernet Kabel
4. IO-Adapter
5. DVD für die Installation des SDK
6. Einem „First Steps“ Guide

Abbildung 2: *leanXcam Kit.*

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

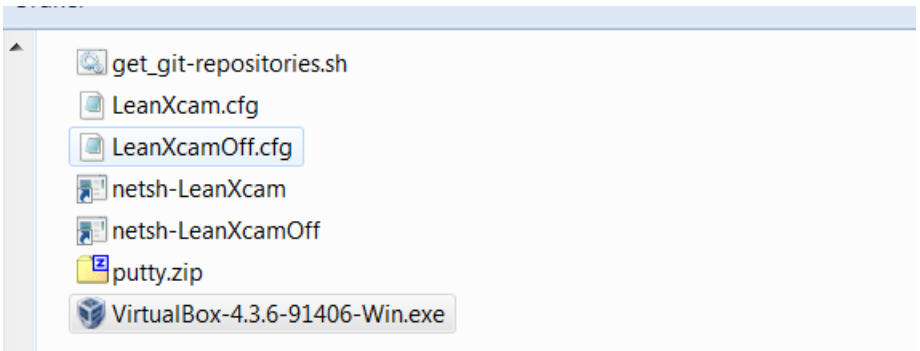


Abbildung 3: *Verzeichnis leanXcam\SW auf ILIAS*

2.2. Allgemeiner Überblick

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

eine virtuelle Maschine (VM)[1]. Dies hat den Vorteil, dass die teilweise nicht 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 leanXcam-HW sowie einem WLAN Interface zum Internet Access.
- **leanXcam**:
  Intelligente Kamera für Bildverarbeitung.
- **Linux VM**:
  Virtuelle Maschine, welche das SDK für die Entwicklung auf der leanXcam beherbergt.

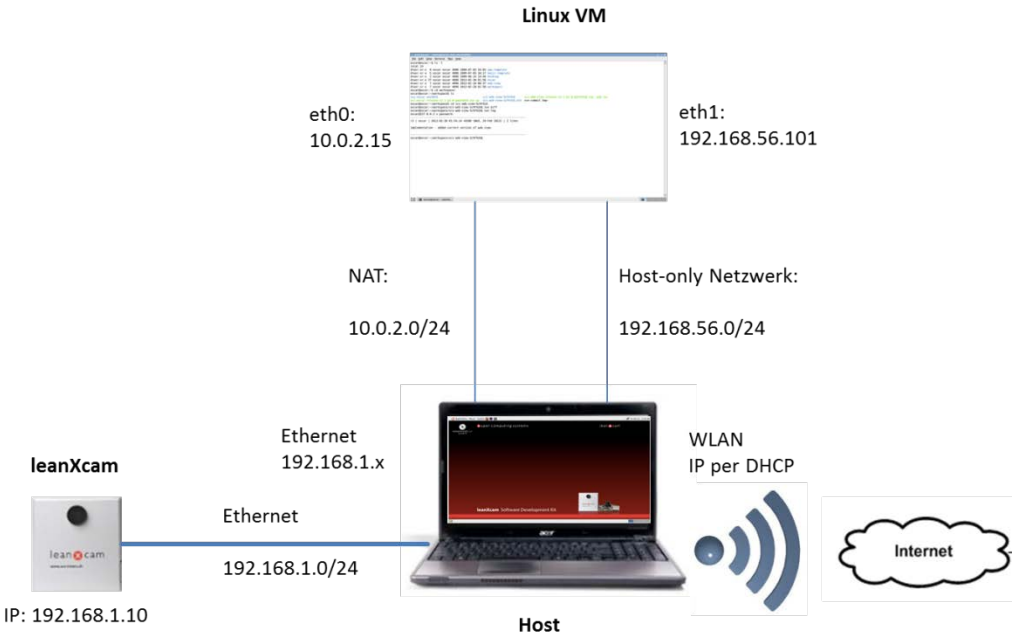


Abbildung 4: *Überblick über die HW- und SW-Komponenten und deren gegenseitige Anbindung.*

In Abbildung 4 sind auch die verschiedenen Netzwerkinterfaces eingezeichnet, die zur Kommunikation zwischen den verschiedenen Komponenten notwendig sind[2]. Details zur Konfiguration finden sich in Abschnitt 7.1. Ziel bei dieser Konfiguration[3] ist, dass eine bidirektionale Kommunikation zwischen Host und VM besteht, als auch, dass von der VM ebenso wie vom Host direkt auf die leanXcam zugegriffen werden kann.

2.3. Installation

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

[1] Die virtuelle Maschine ist eine Ubuntu Linux Distribution (Version 12.04, LTS).
[2] Zum Fileaustausch zwischen Host und VM existiert auch die Option eines „Gemeinsamen Ordners".
[3] Die Default Konfiguration einer VM auf der VirtualBox besteht nur aus dem NAT Interface, ohne die Möglichkeit eines Zugriffes vom Host auf die VM.

2.3.1. Anbindung der leanXcam per Ethernet an den Host

Die leanXcam wird serienmässig mit der IP 192.168.1.10/24[4] ausgeliefert. Die physikalische Anbindung erfolgt über den Ethernet Adapter des Host mit Hilfe des im leanXcam Kit enthaltenen Ethernet Kabels (Abbildung 2, Item 3). Für eine funktionierende IP-Kommunikation zwischen Host und leanXcam 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[5] (mit Ausnahme von 192.168.1.10, welche von der leanXcam besetzt ist) können verwendet werden. Unter Windows[6] können die Netzwerkeinstellungen unter

`Systemsteuerung\Alle Systemsteuerungselemente\Netzwerk- und Freigabecenter`

gemäss Abbildung 5 vorgenommen werden. Es gibt ebenfalls eine kurze Anleitung im Guide „First Steps“ des leanXcam Kits (Abbildung 2, Item 6).

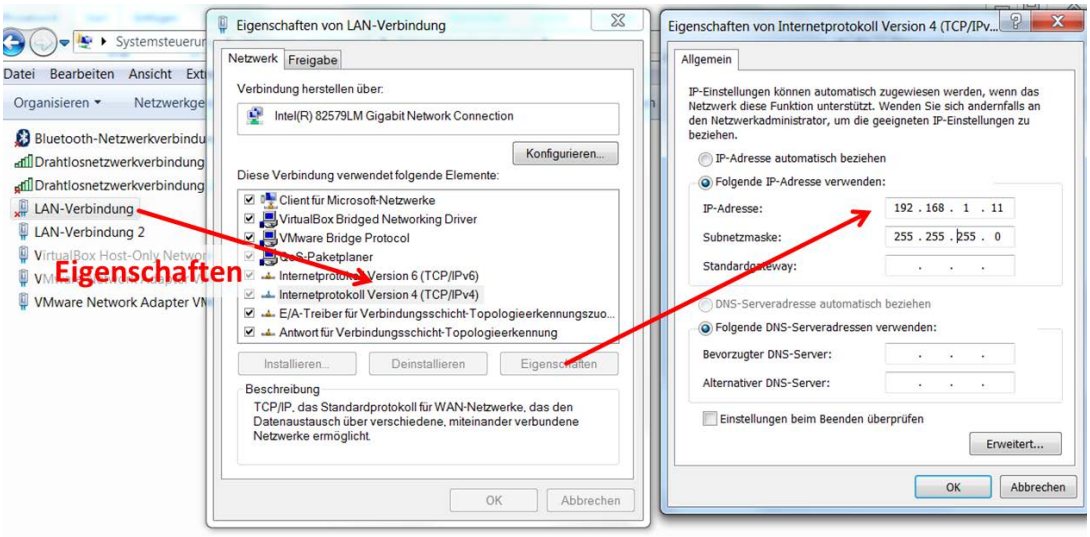


Abbildung 5: *Beispiel für das Setzen der IP-Adresse 192.168.1.11/24.*

Das manuelle Setzen der IP Adresse über die Systemsteuerung von Windows ist recht mühselig, vor allem wenn Sie die Einstellungen häufig ändern müssen. 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-LeanXcam` und `netsh-LeanXcamOff` im SW-Verzeichnis (Abbildung 3) realisiert. Wenn beide Skripte – zusammen mit den zugehörigen Konfigurationsdateien `LeanXcam.cfg` bzw. `LeanXcamOff.cfg` – in das Verzeichnis `C:\leanXcam` kopiert werden, kann durch ausführen des Skripts `netsh-LeanXcam` (als Administrator unter Windows 7) die IP Adresse des Ethernet Interfaces[7] auf den Wert 192.168.1.11/24 gesetzt werden. Ausführen des Skripts `netsh-LeanXcamOff` (als Administrator unter Windows 7) setzt den Wert wieder auf eine automatische IP-Konfiguration per DHCP zurück.

Für einen ersten Test zur Verifikation der Ethernet Verbindung mit der leanXcam können Sie auf der Kommandozeile einen ping-Request schicken (Abbildung 6). Sollte

[4] Für die Bedeutung dieser Schreibwiese siehe [4].
[5] 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.
[6] Für die Betriebssysteme Linux bzw. Mac OS verwenden Sie ggf. die zugehörigen „Hilfe und Support“ Anweisungen.
[7] 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.

die leanXcam bei korrekter Einstellung des Netzwerkes nicht antworten, so hilft in der Regel ein Neustart der leanXcam (bei bestehender Anbindung per Ethernet Kabel)[8].

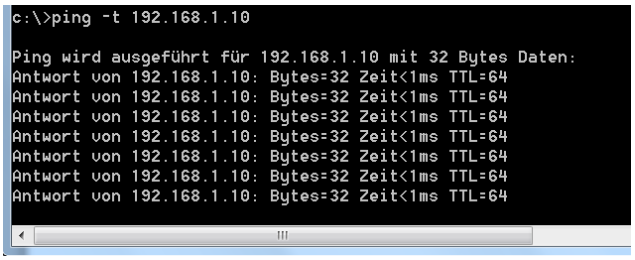


Abbildung 6: *Test der Kommunikation zwischen Host und leanXcam mittels Ping.*

2.3.2. Kommunikation mit der leanXcam via Browser

Auf der leanXcam ist standardmässig eine Applikation installiert, die vom installierten Boa [5] Webserver gesteuert wird. Die Kommunikation vom Host findet via Browser statt. Hierzu einen Webbrowser öffnen und in der Adresszeile die IP-Adresse der leanXcam (192.168.1.10) eingeben (Abbildung 7) und die Enter Taste drücken. Es öffnet sich eine Webseite, auf der verschiedene Einstellungen vorgenommen werden können.

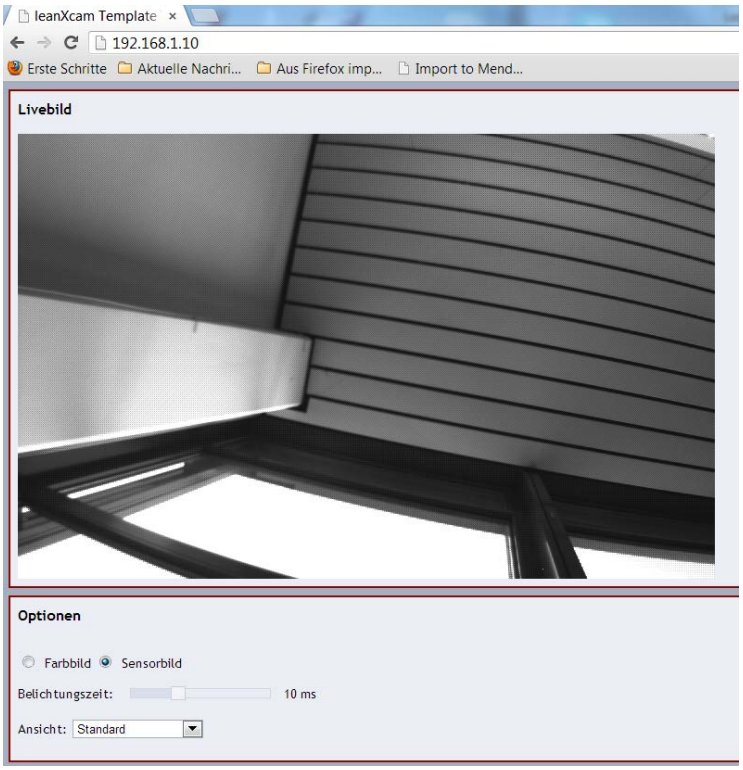


Abbildung 7: *Kommunikation via Browser mit der leanXcam.*

[8] Die folgende Reihenfolge, bei der Installation schein am robustesten: 1) IP-Adresse des Ethernet Netzwerkinterfaces des Host setzen -> 2) Ethernet Kabel verbinden -> 3) Netzkabel der leanXcam verbinden und so die leanXcam starten.

2.3.3. Kommunikation mit der leanXcam via Putty

Putty ist ein u.a. für Windows verfügbarer Client mit dem u.a. eine Secure Shell (SSH) aufgebaut werden kann. SSH ist (neben Telent) das Standardprotokoll für eine sichere Remoteverbindung mit einem Linux Rechner. Daher eignet sich Putty für die Kommunikation mit der leanXcam.
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.

WARNUNG:
Mit einer SSH-Verbindung auf die leanXcam haben Sie dort Root-Rechte (vergleichbar zu Administratorrechten unter Windows) und haben Zugriff auf alle Dateien. Wenn durch eine Fehlkonfiguration das Netzwerkinterface der leanXcam nicht mehr funktionsfähig ist, können Sie die Kamera nicht mehr verwenden. Sie kann dann nur mühsam über eine RS232 Verbindung rekonfiguriert werden. Seien Sie daher bei allen Aktionen, die auf der leanXcam selbst ausgeführt werden[9], eher vorsichtig!

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

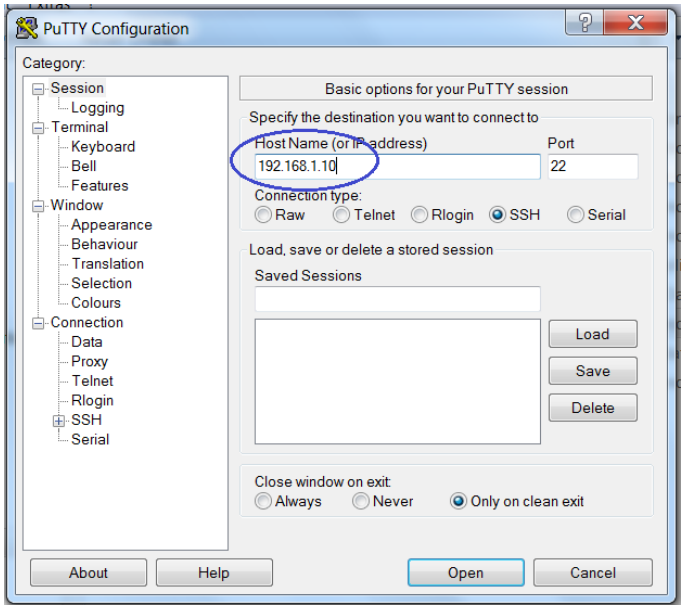


Abbildung 8: *Dialog der putty Anwendung zur Öffnung einer SSH-Verbindung.*

Danach öffnet sich ein Terminal Fenster (Abbildung 9) mit der Aufforderung zur Eingabe des Passwords. Nach dessen Eingabe (`oscar`, in Kleinbuchstaben) steht die SSH-Verbindung zum uClinux Betriebssystem der leanXcam zur Verfügung.

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 Web-Applikation zu erkennen, sowie der Boa Webserver direkt darunter.

[9] Hierzu gehören auch ssh Befehle, die aus Linux Skripten oder Makefiles aus aufgerufen werden.

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

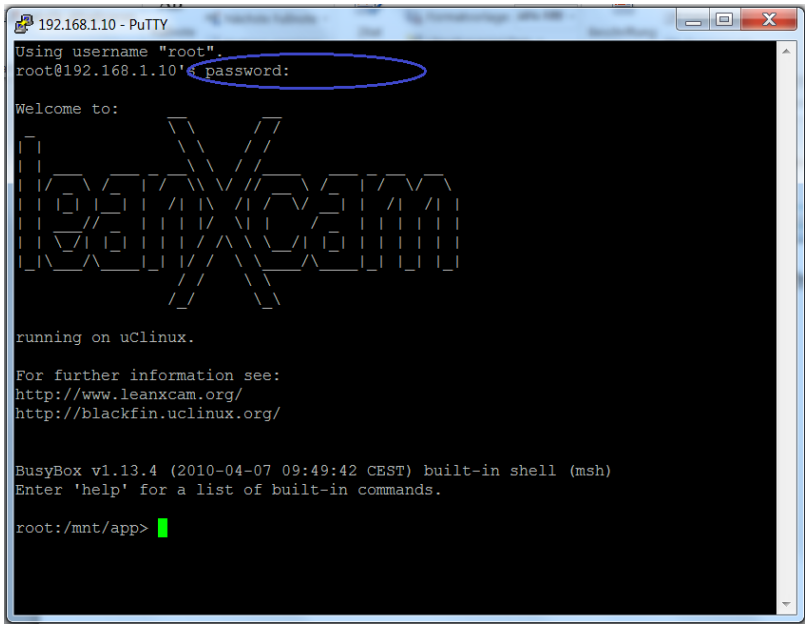


Abbildung 9: *Terminal Fenster einer SSH-Verbindung via Putty zur leanXcam.*

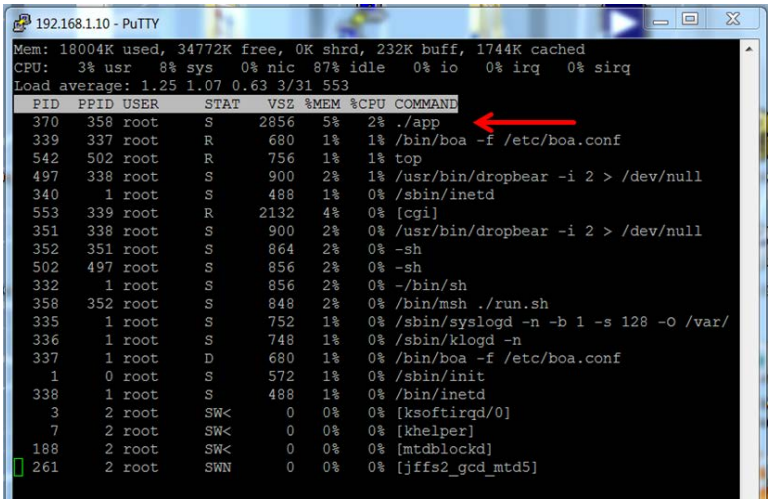


Abbildung 10: *Der Linux Befehl `top` zeigt die laufenden Prozesse an.*

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

3. Installation der VM mit dem SDK

Wie einleitend bereits bemerkt, wird das Software Entwicklungskit (SDK) der leanXcam über eine Virtuelle Maschine (VM), welches eine Ubuntu Linux Distribution (Version 12.04, LTS) enthält, ausgeliefert[10]. Um die VM zu starten wird die Virtualisierungssoftware VirtualBox benötigt. Das Installationspaket für Windows[11] liegt im `leanXcam\SW` Verzeichnis auf ILIAS (Abbildung 3).

Nach der Installation der VirtualBox muss die VM installiert werden. Verwenden Sie hierzu **nicht** die DVD des leanCam Kits (Abbildung 2, Item 5) sondern die im Unterricht per USB-Stick verteilte VM. Letztere beinhaltet schon verschiedene Updates.
Gehen Sie dazu wie in Abbildung 11 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.

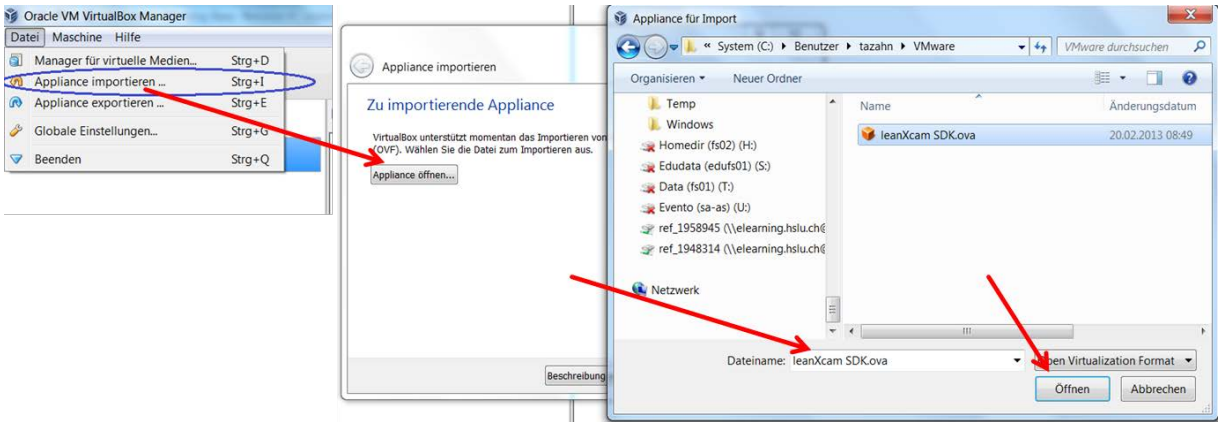


Abbildung 11: *Importieren der Virtuellen Maschine mit dem leanXcam-SDK in VirtualBox.*

Bevor wir die VM starten, definieren wir noch einen „Gemeinsamen Ordner“, der einen einfachen Datenaustausch zwischen VM und Host erlaubt[12]. Dies ist in Abbildung 12 zusammengefasst:

1. Unter `Ändern` wählen Sie `Gemeinsamer Ordner`
2. dann im Dialog rechts auf das „Plus“  Symbol drücken
3. in dem daraufolgenden 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 könnne Sie auf den Pfeil 

drücken und damit die VM starten. Das Ubuntu Betriebssystem fährt dann hoch und der Default Desktop (Abbildung 13) wird sichtbar.

[10] Im Anhang 8.3 findet sich auch eine Anleitung zur Installation des SDK direkt auf einem Linux Betriebssystem.
[11] Für Linux oder Mac finden Sie die Pakte unter [6].
[12] Ansonsten sind VM und Host nur über die Netzwerk Schnittstellen verbunden, was einen Fileaustausch etwas schwerfällig macht.

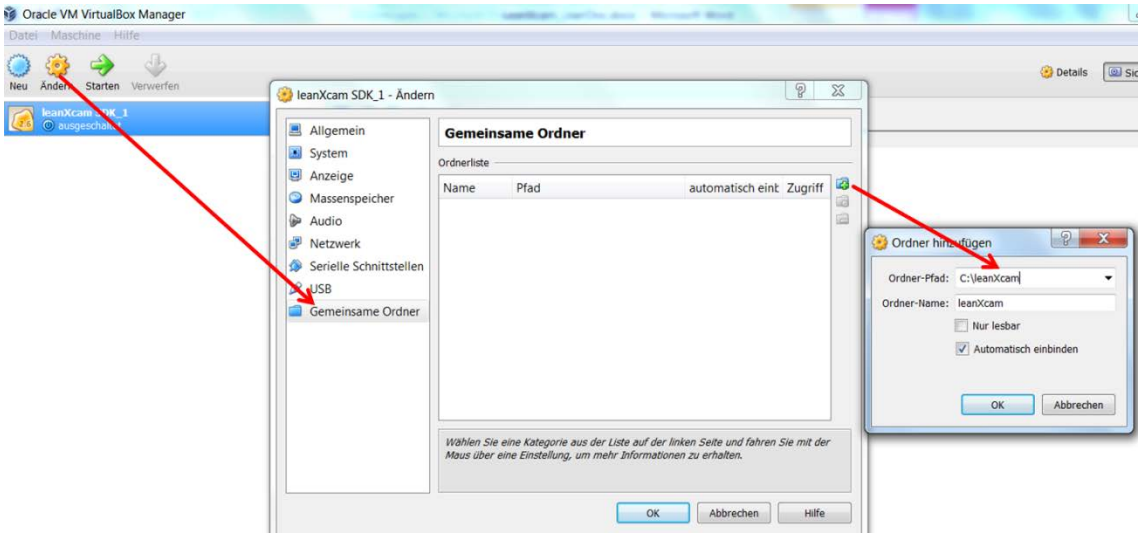


Abbildung 12: *Definition eines Gemeinsamen Ordners zum einfachen Datenaustausch zwischen Host und VM.*

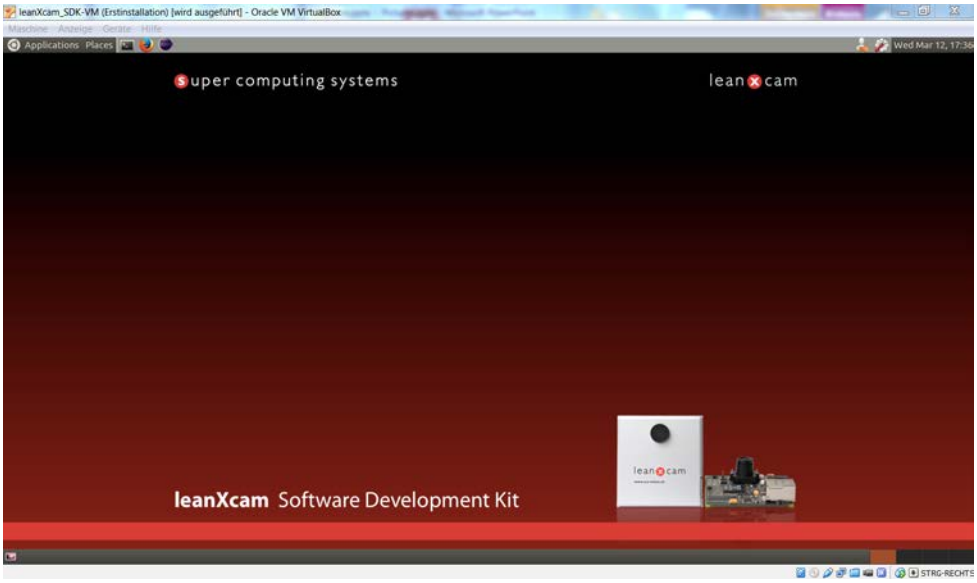


Abbildung 13: *Desktop der Ubuntu VM.*

3.1. Eine 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 leanXcam HW notwendig (Abschnitt 2.3.3).

Ein Kommandozeilenfenster („Terminal“) wird durch Drücken der entsprechenden Taste auf dem Desktop (Abbildung 14) geöffnet.

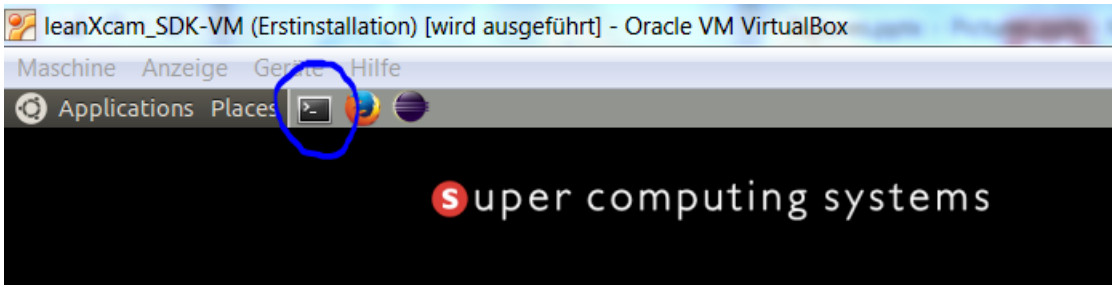


Abbildung 14: *Öffnen eines Terminals (entspricht dem Kommandozeilen Fenster unter Windows) in der leanXcam-SDK VM.*

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 root:leanXcam`: Ändert die Besitzer- und Gruppenzugehörigkeit der angegebenen Datein(en) rekursiv auf `root` bzw `leanXcam`

Prozesses:

- `chmod a+x app`: erteilt allen Benutzern Ausführrechte der Applikation `app`[13]
- `./app`: Started die ausführbare Applikation `app`[14]
- `./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 die leanXcam
- `ifconfig`: Zeigt bzw. ändert die aktuelle Netzwerkkonfiguration
- `sudo /etc/init.d/networking restart`: Neustart d. Netzwerkinterfaces[15]
- `netstat –r`: Zeigt die Routing Tabelle an.
- `route del`: Löscht eine Route
- `route add`: Fügt eine Route hinzu

Makefiles:

- `./configure`: startet das configure Skript[16]
- `make`: erstellt eine Applikation mittels Makefile
- `make clean`: löscht alle Binär- und Objektfiles einer Applikation

[13] Damit ein Programm ausführbar ist, muss es diese Ausführrechte erhalten.
[14] Der Pfad (hier ./ zum aktuellen Verzeichnis) muss immer angegeben werden.
[15] 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 `oscar` (Kleinbuchstaben) abgefragt wird.
[16] Dieser Befehl ist vor der ersten Erstellung einer Applikation mittels `make` auszuführen.

- `make deploy`: Kopiert die Applikation auf die leanXcam
- `make run`: Startet die Applikation auf der leanXcam

Remote:

- `ssh 192.168.1.10`: Stellt eine SSH-Verbindung zur leanXcam 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[17]. Für einen ersten Test kann auf der Konsole eines Terminals (Abbildung 14) der Befehl `ifconfig` ausgeführt werden. Es sollten – wie in Abbildung 15 gezeigt – die beiden Netzwerk Interfaces angezeigt werden, sowie das Local Loopback Interface auf der Adresse 127.0.0.1/8.

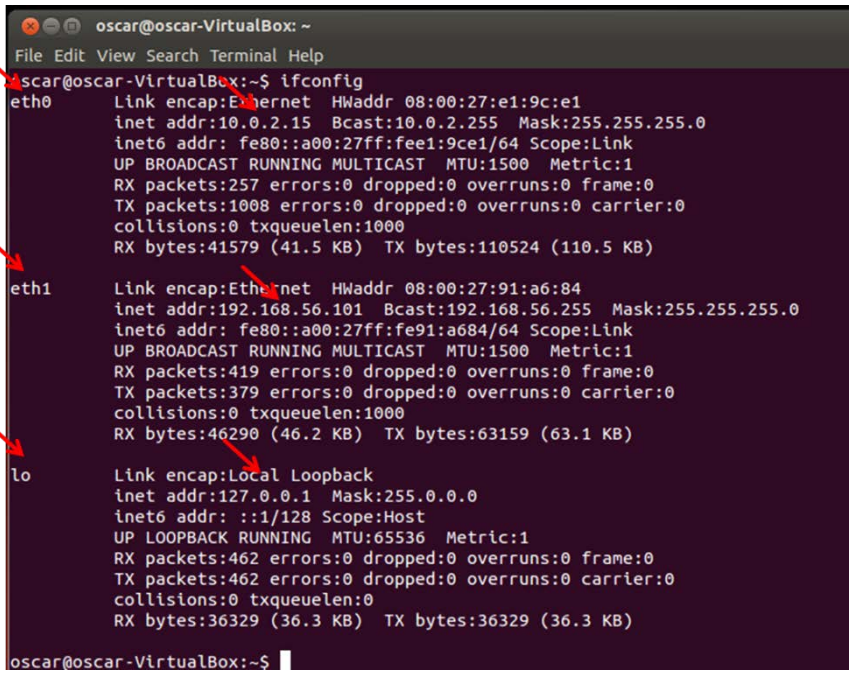


Abbildung 15: *Netzwerk Interface der leanXcam-SDK VM.*

Ggf. ist dann die Routing Tabelle noch nicht korrekt (wie in Abbildung 16 gezeigt) und die Default Route zeigt nicht auf das NAT Interface (IP 10.0.2.2/24 auf `eth6`) sondern noch auf das Host-only Interface (IP 192.168.56.0/24 auf `eth7`). Dann:

1. Mittels `sudo route del default` die falsche Default Route löschen
2. Mittels `sudo /etc/init.d/networking restart` einen Neustart der Netzwerk Interfaces initiieren.

Nun sollte die Routing Tabelle wie in Abbildung 16 gezeigt aussehen und die folgenden Tests alle erfolgreich durchgeführt werden können.

[17] In Kapitel 6.1 sind die Schritte zum Einstellen der Netzwerk Settings der VirtualBox beschrieben.

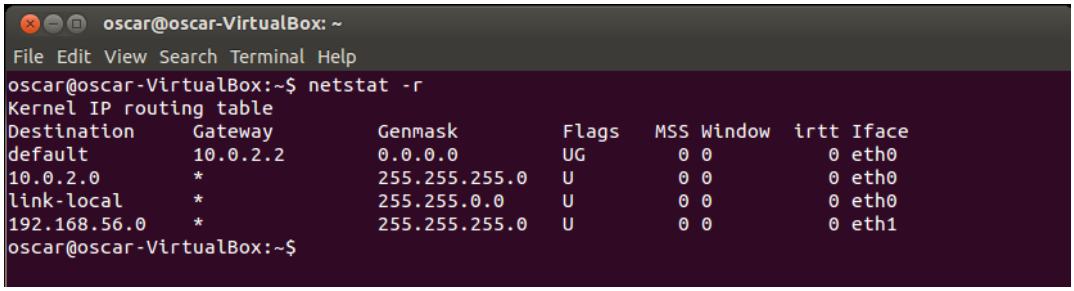


Abbildung 16: *Routing Tabelle der leanXcam-SDK VM.*

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

1. Öffnen des Mozilla Browsers zum Test für den Internet Zugriff.

2. `ssh 192.168.1.10`:
   Ebenso wie der Host (s. Abschnitt 2.3.3) kann auch die leanXcam-SDK VM eine SSH-Verbindung zur leanXcam 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 `oscar` (Kleinbuchstaben) wird ein Terminal zur Remoteverbindung zur leanXcam 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 leanXcam-SDK VM definiert, womit dessen Position klar ist (Abbildung 12).
In der leanXcam-SDK VM ist der Ordner im Pfad `/media/xxx` zu finden, wobei `xxx` dem Namen des Verzeichnisses im Host entspricht (ebenfalls im Konfigurationsdialog (Abbildung 12) angezeigt). Zum Test öffnen Sie in der VM unter `Places -> Home Folder` einen File Browser (Abbildung 17) und zeigen Sie via `File System` 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 leanXcam-SDK VM) und die Änderung auf der jeweils „anderen Seite“ verifizieren.

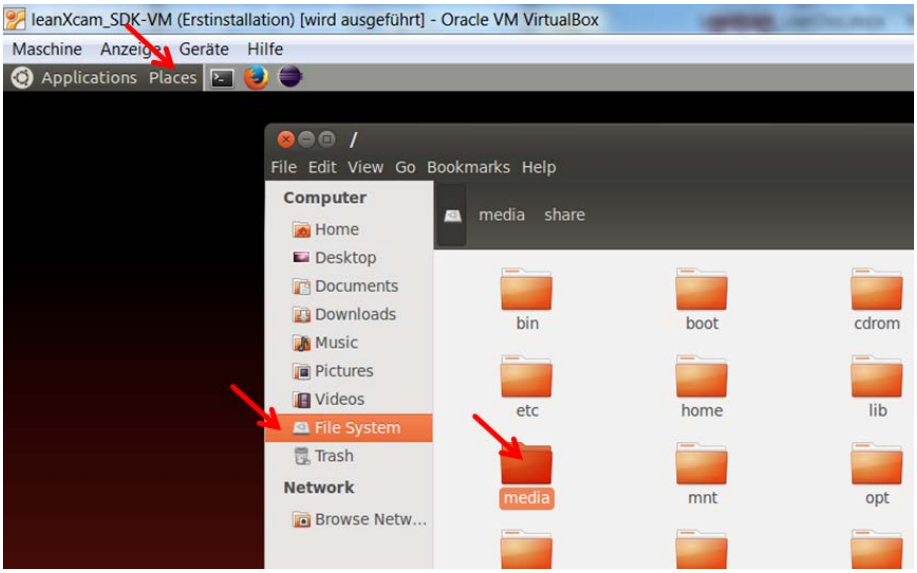


Abbildung 17: *Aufsuchen des "Gemeinsamen Ordners" in der VM.*

3.2.3. Kommunikation von Host zu VM via SSH

Eine SSH Verbindung vom Host zur leanXcam-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 leanXcam 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 leanXcam-SDK VM via der IP 192.168.56.101 hergestellt. Dabei ist auch wieder das Passwort `oscar` (Kleinbuchstaben) einzugeben.

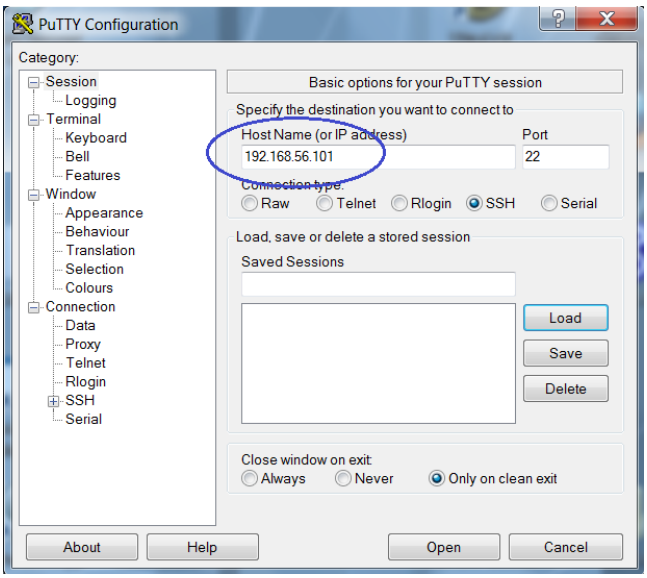


Abbildung 18: *Herstellen einer SSH-Verbindung vom Host zur leanXcam-SDK VM.*

4. Verwendung des leanXcam SDK

Die Verwendung des SDK ist mittels zahlreicher Beispielprogramme sehr gut vorbereitet und es wäre prinzipiell möglich, direkt in die Entwicklung von Bildverarbeitungsanwendungen einzusteigen. Damit aber die grundsätzlichen Zusammenhänge der verschiedenen Komponenten klar werden, 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.

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 leanXcam-SDK VM in das home-Verzeichnis des oscar Users[18]:

   ```
   cd ~
   ```

   Verifizieren Sie mittels

   ```
   pwd
   ```

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

   ```
   cd uebung
   ```

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

   ```
   mkdir uebung01
   ```

[18] Die Tilde ~ ist ein Shortcut für das Home-Verzeichnis.

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, oder aber selbst eine Anwendung entwickeln:

   ```
   #include <stdio.h>

   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

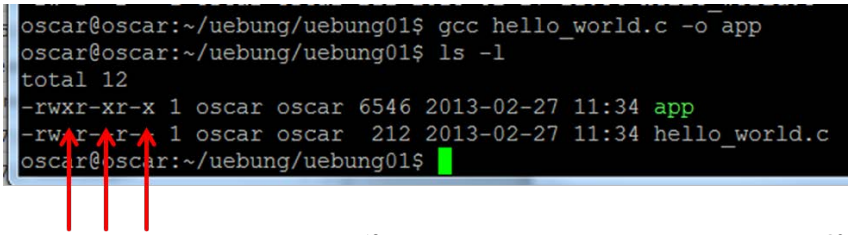
   ```
   gcc hello_world.c –o app
   ```

   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:

   Das Flag x (für ausführbar[19]) ist für alle drei Benutzerklassen[20] (user/group/others) gesetzt[21].

6. Führen Sie das Programm aus mittels:

[19] ‘r‘ bzw. ‚w‘ stehen für Lese- (read) und Schreib- (write) Rechte.
[20] Für Details siehe hierzu http://de.wikipedia.org/wiki/Unix-Dateirechte#Benutzerklassen.
[21] Um das Flag ‚x‘ manuell zu setzen, verwenden Sie `chmod a+x Filename`.

```
./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 uclinux in C

Ziele:

1. Wir können ein C-File kompilieren und linken, um einen ausführbaren Code für das uclinux Betriebssystem zu erzeugen.

2. Wir können den Binärcode für uclinux auf die leanXcam übertragen und ausführen.

Übung im Detail:

1. Erstellen Sie ein neues Verzeichnis `uebung02` unter `/home/oscar/uebung` und wechseln Sie in das Verzeichnis.

2. Kopieren Sie das File `hello-world.c` (aus Abschnitt 4.1.1) in dieses Verzeichnis[22]:

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

   Verifizieren Sie das Ergebnis.

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

   ```
   bfin-uclinux-gcc -elf2flt="-s 1048576" hello_world.c -o app
   ```

   Dabei legt die Anweisung `-elf2flt` das Format des Binar Codes fest. Das Standardformat auf Linux Betriebssystemen ist ELF (Executable and Linking Format). Das uclinux Betriebssystem auf der leanXcam unterstützt (nur) das kompaktere FLAT Format. Weiterhin wird die maximale Stack Grösse des Programmes auf 1 MB (`="-s 1048576"`) festgelegt.

4. Verifizieren Sie wieder die Zugriffsrechte auf des Binärfiles (`app`). Dabei werden Sie feststellen, dass neben der Datei `app` eine weitere Datei `app.gdb` erzeugt wird. Diese ist für Debugging Zwecke und werden wir später noch kennenlernen.

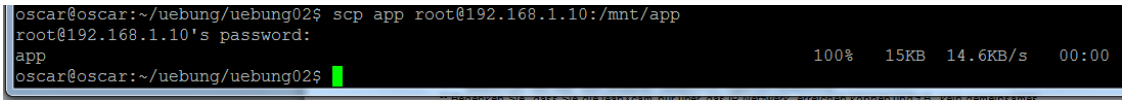
5. Übertragen Sie nun das Binärfile (`app`) auf die leanXcam. Dies erfolgt mittels der Applikation `scp`:[23]

[22] 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.

[23] Bedenken Sie, dass Sie die leanXcam nur über das IP-Netzwerk erreichen können und z.B. kein gemeinsames Verzeichnis zum Dateiaustausch haben.

```
scp app root@192.168.1.10:/mnt/app
```

Zur Syntax: scp überträgt das File `app` (im lokalen Verzeichnis[24]) 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 `root` und kopiert die Datei dort in das Verzeichnis `/mnt/app`. Nach dem Ausführen des Befehls, werden Sie nach dem Passwort des Benutzers `root` auf dem Remote Computer `192.168.1.10` gefragt. Dieses ist – wie für die SSH-Verbindung auf die leanXcam (Kapitel 2.3.3) schon verwendet – `oscar` (Kleinbuchstaben). Es sollte eine Status Meldung wie folgt angezeigt werden:



6. Verbinden Sie sich nun per SSH auf die leanXcam:

   ```
   ssh root@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 `/mnt/app` vorhanden ist und starten Sie das Programm.

7. Löschen Sie die Datei `app` auf der leanXcam mittels:

   ```
   rm app
   ```

   Dann schliessen Sie am Ende die SSH-Verbindung wieder.

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

Ziele:

1. Wir kennen den Unterschied zwischen gcc und g++.

Übung im Detail:

1. Erstellen Sie ein neues Verzeichnis `uebung03` unter `/home/oscar/uebung` und wechseln Sie in das Verzeichnis.

2. Erzeugen Sie mit Hilfe eines Editors (`gedit`) ein kleines C++-Quell File `hello_world.cpp`. Im Folgenden wieder ein kleines Beispiel, welches Sie direkt übernehmen können, aber Sie können auch selbst eine kleine Anwendung entwickeln:

[24] scp kann auch zum Filetransfer zwischen zwei verschiedenen Remote Computern verwendet werden.

```
#include <stdio.h>

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

  int a;
  int b;
};

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 uclinux Betriebssystem mittels

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

   bzw.

   ```
   bfin-uclinux-g++ -elf2flt="-s 1048576" 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 leanXcam-SDK VM als auch auf der leanXcam selbst (analog 4.1.1 bzw. 4.1.2).

4.1.4. Übung 4: Einbindung einer externen Library/Skripting

Ziele:

1. Wir kennen den Unterschied zwischen dem Kompilieren und Linken und dem Bezug zu `#include` Statement sowie dem Einbinden von externen Libraries.

2. Wir können externe Libraries in ein Programm einbinden.

3. Wir können ein einfaches Shell-Skript erstellen.

Übung im Detail:

1. Erstellen Sie ein neues Verzeichnis `uebung04` unter `/home/oscar/uebung` und wechseln Sie in das Verzeichnis.

2. Erzeugen Sie mit Hilfe eines Editors (`gedit`) das folgende C-Source File `hello_math.c`:

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

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

  printf("hello C-world\n");
  printf("cosine(3.14) = %0.5f\n", cos(3.14));
  return(0);
}
```

3. Studieren Sie kurz den Code und beachten Sie vor allem das

   ```
   #include <math.h>
   ```

   Statement. Wieso braucht es dieses?

4. Wir werden nun die Erstellung des Binärcodes in zwei Schritte unterteilen:

   a. Wir kompilieren die Source Files und erstellen ein sog. Objektfile. Dieses stellt noch keinen ausführbaren Code dar:

      ```
      bfin-uclinux-gcc –c hello_math.c
      ```

      Das Flag `–c` weisst 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, 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:

      ```
      bfin-uclinux-gcc -elf2flt="-s 1048576" hello_math.o –o app
      ```

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

      ```
      hello_math.c:(.text+0x27): undefined reference to `cos'
      ```

      Können Sie sich denken, woher der Fehler stammt?
      Das `#include <math.h>` Statement weisst den Compiler auf die

Definition der Funktion `cos()` hin, d.h. im Kompilier-Schritt kann der Compiler die korrekte Verwendung der Funktion bez. Syntax und Typ überprüfen. Aber die eigentliche Implementierung, liegt dem Compiler noch nicht vor. Diese bracht er auch erst beim Linken, dann, wenn er den ausführbaren Code erzeugen will.
Daher müssen wir beim Linken noch das Objektfile mit der Implementierung der `cos()` Funktion angeben. Meist sind solche externen Objektfiles in Bibliotheken (Libraries) zusammengefasst, die mit der Dateiendung `*.a` gekennzeichnet sind. Die math-Library heisst `libm.a` und für das Einbinden genügt der Aufruf `-lm`. Dabei ist die Konvention, dass dies eine Abkürzung für libm.a ist und die Library sich in einem der Suchpfade des Compilers befindet:

```
bfin-uclinux-gcc -elf2flt="-s 1048576" hello_math.o –lm
-o app
```

5. Verifizieren Sie wieder die Funktionalität des Programmes.

   Frage: Rechnet der Kosinus in Grad oder im Bogenmass?

6. 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 ganz einfache Einführung sei folgendes Shell Skript gedacht:

   ```
   #!/bin/bash

   echo "compile $1"
   bfin-uclinux-gcc -c $1.c

   echo "link $1 to file app"
   bfin-uclinux-gcc -elf2flt="-s 1048576" $1.o -lm -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.

Nachdem wir nun einen ersten Einstieg in die Programmierung von C(++) unter Linux für die leanXcam geschafft haben, wenden wir uns nun unserem eigentlichen Interesse an der leanXcam zu, nämlich der Bildverarbeitung. Hierzu müssen wir uns mit einer speziell für die leanXcam entwickelten Bildverarbeitungsbibliothek vertraut machen, dem sog, Oscar Framework.

4.2. Das OSCar-Framework

Das Oscar Framework hat die folgende Philosophie [7]:

„*Um den Einstieg in die Verwendung der leanXcam zu vereinfachen und die Entwicklungsarbeit möglichst gering zu halten, wird das OSCar (Open-Source Camera) Software-Framework mitgeliefert, welches die Hardware der leanXcam abstrahiert und dem Entwickler einfache Funktionen zu deren Benutzung zur Verfügung stellt. Diese Programmbibliothek ist ebenfalls mitsamt dem C-Quellcode frei erhältlich (LGPL Lizenzmodell) und wird zusammen mit der Community stetig erweitert, damit, ganz nach dem Open-Source-Gedanken, die Arbeit eines Teams allen zugutekommt.*“

4.2.1. Installation des Frameworks

Das Framework ist bereits in der leanXcam-SDK VM installiert. Daher können Sie diesen Abschnitt überspringen. Er ist nur dann relevant, wenn das OSCar Framework neu installiert werden soll.

1. Download des Frameworks vom git-Repository[25] ins aktuelle Verzeichnis:

   ```
   git clone git://github.com/scs/oscar.git
   ```

   Wechseln Sie danach in das `oscar` Verzeichnis.

2. Zur Erstellung verschiedener Konfiguration (z.B. HW-Plattformen) dient das `configure` Skript. Es werden dann verschiedene Optionen angeboten, die auf der Kommandozeile ausgewählt werden können.

   ```
   ./configure
   ```

   Hier ist lediglich die HW-Plattform `[leanXcam]` zu wählen.

3. Erstellen der Bibliotheken:

   ```
   make
   ```

   Make ist ein sehr mächtiges Build Management Tool, welches zur Entwicklung von SW unter Linux verwendet wird (s. Kapitel 5).

4. Erstellen der Dokumentation[26]:

   ```
   doxygen documentation/oscar.doxygen
   ```

[25] git ist ein Versionierungstool, welches für die Verwaltung der leanXcam SDK Quelldateien verwendet wird. Ein Tutorial findet sich unter [8].

[26] Doxygen [9] ist ein de-facto Standard für die automatisierte Erstellung von SW-Dokumentation aus dem C(++) Quellcode. Der immense Vorteil dieser Methode ist, dass die Dokumentation wirklich aktuell ist. Einzige Voraussetzung ist, bestimmte Konventionen bei der Quellcode Kommentierung einzuhalten.

Dieser Befehl erstellt eine html-Dokumentation (Default), wobei der Einstieg das File

`documentation/html/index.html`

ist.

Damit ist die Installation schon abgeschlossen.

4.2.2. Dokumentation

Die automatisch per Doxygen erstellte Dokumentation (s. Abschnitt 4.2.1) finden Sie entweder auf ILIAS (`leanXcam\documentation`) oder über eine entsprechende Bookmark im Firefox Browser der VM:

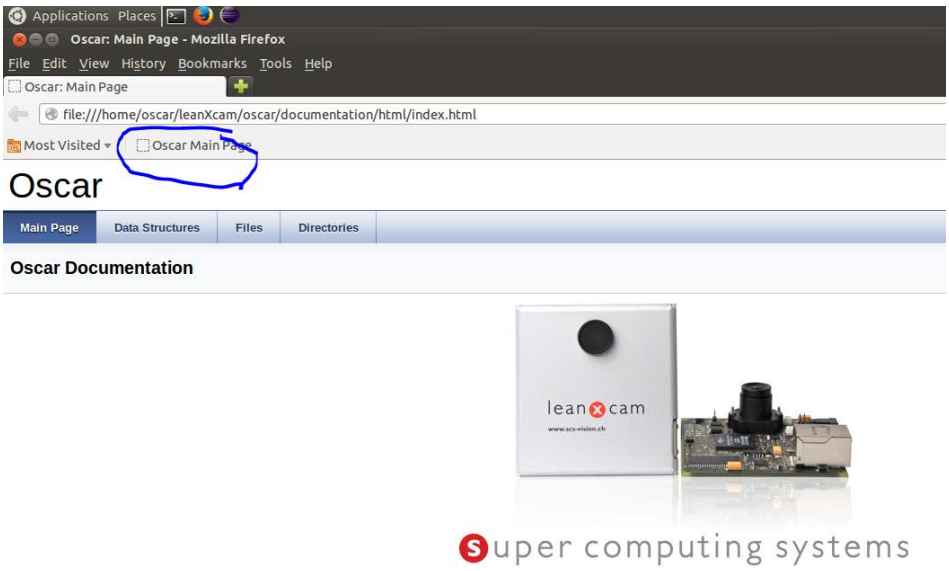


Abbildung 19: *Die Dokumentation für das OSCar Framework findet sich direkt über eine Bookmark im Browser der leanXcam-SDK VM.*

4.2.3. Wesentliche Komponenten des OSCar Frameworks

Das OSCar Framework übernimmt die wesentlichen Funktionalitäten im Zusammenhang mit der Verwendung der leanXcam HW. Es ist allerdings keine Library für Bildverarbeitung (wie openCV[27]) auch wenn im Vision Library Modul (vis.h) einige elementare Filterfunktionen implementiert sind.
Im Folgenden, sind einige Links[28] von der Seite `include Directory Reference` der Dokumentation.

file	**bmp.h** [code] *Lesen und Schreiben von Bitmaps.*
file	**cam.h** [code] *Akquisition von Bildern und Kamera Einstellung.*
file	**dma.h** [code]

[27] Es gibt auch eine Version der openCV für die leanXcam auf [10].
[28] Wenn das ILIAS Verzeichnis `leanXcam\documentation` im selben Verzeichnis liegt wie das vorliegende Dokument sollten die (relativen) Links direkt funktionieren.

	Aufsetzen einer DMA Chain für zeitkritische Anwendungen.
file	**frd.h** [code] *Lesen von Bildern bzw. Bildsequenzen zur Simulation bzw. dem Test von Programmen auf dem Host.*
file	**gpio.h** [code] *Verwendung der GPIO.*
file	**hsm.h** [code] *Hierarchische **State** Maschine.*
file	**ipc.h** [code] *Interprozess Kommunikation.*
file	**jpg.h** [code] *API definition for jpeg module.*
file	**log.h** [code] *Logging.*
file	**vis.h** [code] *Vision Library für verschiedene Filteroperationen.*

4.2.4. Einbindung des OSCar Frameworks

Die Einbindung in eigene C(++) Programme erfolgt als externe Library (analog Übung04, 4.1.4). Dabei ist zu beachten:

1. Include Pfad: `../oscar/include`

2. Notwendige Präcompiler Direktiven[29]:

   Eine der drei folgenden ist je nach Anwendung zu selektieren
   `-DOSC_HOST`: Erzeugt Code für Linux Host
   `-DOSC_TARGET`: Erzeugt Code für leanXcam Target
   `-DOSC_SIM`: Erzeugt Code für eine Target Simulation

   `-std=gnu99`: Verwendung des ISO C Standard aus dem Jahre 1999 [11]

3. Library Pfad: `../oscar/library`

4. Libraries:
   Diese sind passend zu den Präcompiler Direktiven oben zu wählen. Die Versionen mit der Endung `_dbg` wurden ohne die Compiler Option `-O2` (Codeoptimierung auf Geschwindigkeit) erstellt und für Debugging Zwecke.

   ```
   libosc_host.a            libosc_host_dbg.a
   libosc_target.a          libosc_target_dbg.a
   libosc_target_sim.a      libosc_target_sim_dbg.a
   ```

[29] Diese sind beim Kompilieren (`gcc -c ..`) anzugeben.

4.3. OSCar Funktionsaufruf

Etwas gewöhnungsbedürftig sind am Anfang die etwas eigenartig anmutenden Funktionsaufrufe in den Beispielen des OSCar Frameworks. Um z.B. die Funktion

```
OscCamSetupPerspective(OSC_CAM_PERSPECTIVE_DEFAULT);
```

aufzurufen, lautet der Code typisch folgendermassen:

```
OscCall( OscCamSetupPerspective, OSC_CAM_PERSPECTIVE_DEFAULT);
```

Dabei wird das MACRO `OscCall` verwendet, wobei dessen erstes Argument der Funktionsname und die weiteren Argumente des Funktionsaufrufes darstellen, so dass die Verwendung prinzipiell sehr einfach ist. Das MACRO `OscCall` definiert ein ausgefeiltes Errorhandling der aufgerufenen Funktion, dessen genaue Definition Sie im File `oscar/include/error.h` nachschauen können. Für eigenen Code können Sie auch die (klassische) obere Versionen verwenden, allerdings profitieren Sie nicht vom Errorhandling.

4.4. Anwendungsbeispiele für das OSCar Framework

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

4.4.1. Übung 5: Einbindung des OSCar Frameworks

Ziele:

1. Wir verstehen erste grundlegende SW-Komponenten einer OSCar-basierten Anwendung.

2. Wir können das OSCar Framework in ein eigenes Programm einbinden.

3. Wir können Daten von der leanXcam auf den Host übertragen.

Übung im Detail:

1. Erstellen Sie ein neues Verzeichnis `uebung05` unter `/home/oscar/uebung` und wechseln Sie in das Verzeichnis.

2. Unter `/home/oscar/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[30]

   ```
   man tar
   ```

[30] Grundsätzlich kann unter Linux mit dem Befehl `man Befehlsname` (Manual Page) oder mittels `Befehlsname` –help Hilfe zu einem Befehl erhalten werden,

(unter Examples) mit der grundsätzlichen Verwendung von `tar` vertraut.

3. Studieren Sie das C-Quellcode File `main.c`. Vollziehen Sie – ggf. mit Hilfe der OSCar-Dokumentation – die grundsätzliche Funktionsweise des Programmes nach.

4. Studieren Sie das Shell Skript `cl.sh`, welches für die Erstellung des Binärcodes verwendet werden kann. Machen Sie sich insbesondere mit folgenden Compiler- bzw. Linker Optionen vertraut:

   `-I../../leanXcamSDK/oscar/include`:
   Definition eines include Verzeichnisses, hier für das OSCar Framework.

   `-DOSC_TARGET`: s. Abschnitt 4.2.4

   `-O2`: Optimiert den Code auf Geschwindigkeit

   Analysieren Sie weiterhin, wie das OSCar Framework beim Linken eingebunden wird.

5. Erstellen Sie nun die Applikation und übertragen Sie auf die leanXcam. Verbindgen Sie sich mit SSH auf die leanXcam und überprüfen Sie mittels

   ```
   top
   ```

   ob die vorinstallierte Applikation `./app` noch ausgeführt wird (s. Abbildung 10). Da immer nur eine Applikation simultan auf die Bilddaten zugreifen kann, müssen wir diese Applikation beenden. Unter Linux erfolgt dies durch

   ```
   killall app
   ```

   oder[31]

   ```
   kill 370
   ```

6. Starten Sie die Applikation auf der leanXcam und beobachten Sie die Konsolen und den Inhalt des Verzeichnisses `/home/httpd`. In letzterem sollte nun eine Serie von Bildern enthalten sein.

7. Nun übertragen Sie die Bilder von der leanXcam auf den Host. Dies machen Sie vom **Host** aus! Erinnern Sie sich an die Verwendung von scp[32]:

   ```
   scp root@192.168.1.10:/home/httpd/out* .
   ```

   Auch die Quelle kann ein Remote Computer sind. Dabei verwenden wir noch den „Wildcard“ `*`, um alle Files auf einmal zu übertragen.

Damit haben wir unsere erste wirkliche Applikation im Zusammenhang mit Bildverarbeitung auf der leanXcam erstellt!

[31] Die Zahl 370 ist die PID (Process-ID). Sie finden die PID eines Prozesses in der ersten Spalte vor dem Prozessnamen in der Ausgabe von `top` (Abbildung 10).
[32] Übersehen Sie nicht den Punkt am Ende des Befehls für die Angabe des Zielverzeichnisses.

Wenn Sie das Programm aufmerksam studiert haben, wird Ihnen sicherlich auch aufgefallen sein, dass Sie nach der Akquisition des Bildes direkten Zugriff auf die Bildrohdaten haben, an der Stelle:

(`main.c`, Zeile 73) `data.pictureRaw.data = pCurRawImg;`

Sie könnten an dieser Stelle schon einmal versuchen, die Bilddaten vor dem Abspeichern zu manipulieren[33].

4.4.2. Übung 6: Verwendung des Boa Webservers

Mit der vorigen Übung (4.4.1) haben wir unser Ziel einer Bildverarbeitung auf der leanXcam Plattform eigentlich schon fast erreicht. Allerdings ist die Visualisierung der Bilddaten noch recht mühsam und sehr langsam mittels `scp` geregelt. 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 von der leanXcam verstehen.

Ziele:

1. Wir verstehen die grundlegende Arbeitsweise des Boa Webservers.

2. Wir können Bilddaten über den Webserver herunterladen.

Übung im Detail:

1. Erstellen Sie ein neues Verzeichnis `uebung06` unter `/home/oscar/uebung` und wechseln Sie in das Verzeichnis.

2. Auf ILIAS finden Sie unter `leanXcam\uebung` das File `uebung06.tar`. Holen Sie sich dieses File und legen Sie es im o.g. Verzeichnis ab. Entpacken Sie das tar-Archiv.
   Es enthält im Wesentlichen eine Kopie der Übung 5 und ein html-File:

   `index.html`

3. Studieren Sie die Änderungen im Skript `cl.sh`: Der Filetransfer wird nach dem Erstellen des Binärcodes auch über das Skript ausgeführt. Zusätzlich zur Applikation wird auch das html-File `index.html` und zwei Textfiles übertragen. Sie liegen im selben Verzeichnis wie die abgespeicherten Bilder:

   `/home/httpd/`

4. 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>
   ```

   und einer Liste von Bildern:

   ```
   <img src="out01.bmp" height="240" width="376">
   <img src="out02.bmp" height="240" width="376"><br>
   ...
   ```

[33] Allerdings müssen die Bilddaten, welche durch ein Bayer Farbfilter ein spezifisches Muster erhalten haben, noch mittels einer Debayer Funktion transformiert werden.

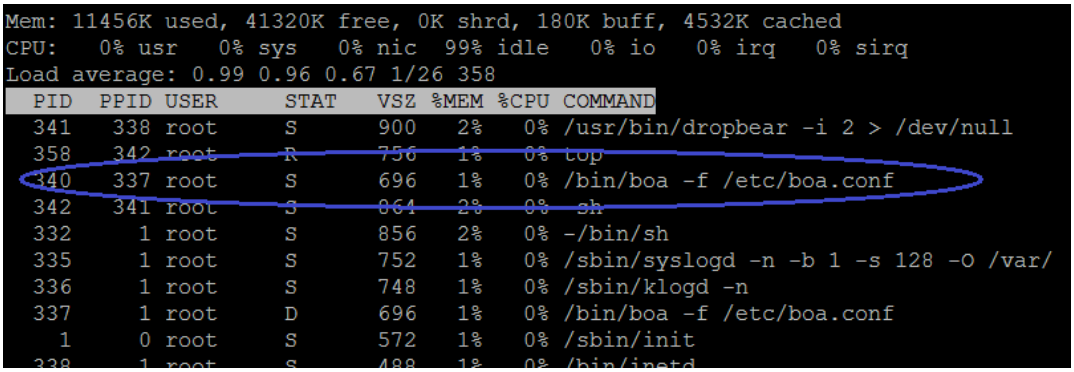
Wie Ihnen sicher auffällt, entsprechen die Bildnamen gerade den abgespeicherten Bildern, welche im selben Verzeichnis liegen wie das html-File[34]. Zwei Beispiel-Textfiles sind ebenfalls im Ordner enthalten.

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.

5. Verbinden Sie sich nun mit SSH auf die leanXcam und geben Sie das Kommando `top` ein[35]. Unter anderem läuft der Prozess `boa` , ein schlanker Webserver speziell auch für embedded Anwendungen [5].



6. Wechseln Sie in das Verzeichnis

   `/home/httpd/`

   und verifizieren Sie die Präsenz des html-Files index.html. Löschen Sie etwaige noch vorhandene Bilder der letzten Tests.

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

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

   Dabei können Sie mit den Pfeiltasten ↑↓ navigieren. Verschiedene Einträge sind interessant:

   a. `Port 80`:
      Wie jeder „anständige“ Webserver „horcht“ auch `boa` auf Port 80.

   b. `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.

   c. `DocumentRoot /home/httpd`
      Das Root Verzeichnis der html Dokumente, die vom Webserver

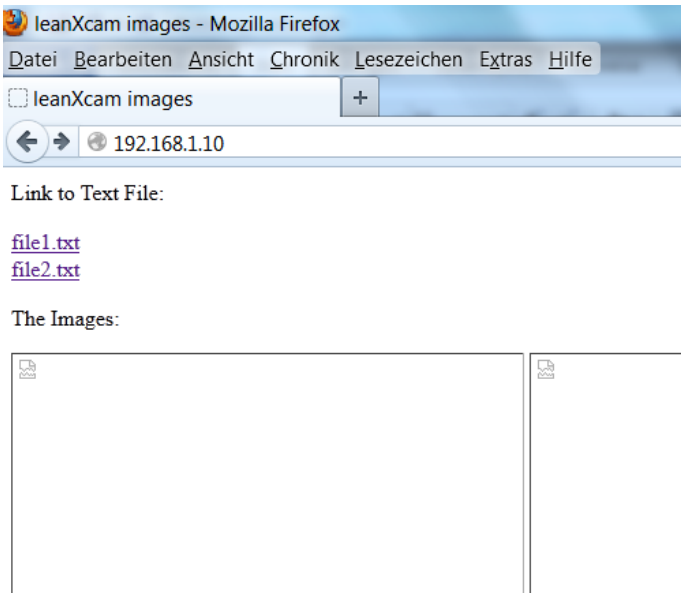
[34] Andernfalls müsste der Pfad relativ zum Verzeichnis des html-Files angegeben werden.
[35] Geben Sie Ctrl-C ein, um die `top`-Anzeige zu verlassen.

veröffentlicht werden.

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

In unserem – sehr einfachen Fall – enthält das File `index.html` nur den Link auf die Textfile und die Liste der Bilder, die von der Applikation abgespeichert werden.

8. Öffnen Sie nun einen Browser und verbinden Sie sich auf die leanXcam (analog Abbildung 7). So lange die Applikation nicht gestartet wurde, liegen noch keine Bilder im Verzeichnis und die Webseite ist praktisch leer.



Allerdings können Sie den Inhalt der Textfiles im Browser anzeigen oder auf der Festplatte speichern.
Starten Sie nun die Applikation auf der leanXcam[36] und beobachten Sie das Verhalten der Webseite. Sie können den Refresh auch manuell ausführen.

9. Falls Sie schon mit den Themen vertraut sind, so können Sie die Applikation erweitern, so dass neben den Bildern eine bestimmte Zeile/Spalte in die Textfiles abgespeichert wird und per Webserver an den Host übertragen werden kann. Stellen Sie das Ergebnis graphisch dar.

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 mittels einer Interprozess Kommunikation (IPC) synchronisiert werden.

[36] Nachdem Sie eine ggf. noch laufende `./app` Applikation mit `killall app` beendet haben.

4.5. Interprozess Kommunikation

Für die Synchronisation wird im Falle der leanXcam ein sog. cgi-Skript verwendet. Einfach gesagt ist ein cgi-Skript nichts weiter als ein ausführbares Programm, welches – durch einen Browser Request getriggert – vom Webserver ausgeführt wird. Das cgi-Skript kommuniziert dann in der Regel über TCP/IP mit der Applikation. Dies ist in Abbildung 20 veranschaulicht.

Die Abfolge eines cgi-Requests läuft wie folgt:

1. Der Browser sendet einen http GET oder POST Request, wobei das cgi-Skript über die URL aufgerufen wird. Dabei können auch Parameter mitgeliefert werden, was besonders beim GET Request – dort sind die Parameter mit einem „?“ an die URL angehängt – sehr einfach ist.

2. Der Webserver startet das cgi-Skript mit den übergebenen Parametern.

3. Das cgi-Skript wird ausgeführt und stellt eine Kommunikation mit der Anwendung her. Z.B. kann das Abspeichern eines Bildes unter einer gewünschten Adresse (Filename) oder die Übertragung per TCP/IP an das cgi-Skript ausgelöst werden.

4. Nachdem das cgi-Skript beendet ist[37], liefert es über stdout das Ergebnis der Verarbeitung[38] an den Webserver zurück[39].

5. Der Webserver liefert das Ergebnis an den Browser zurück.

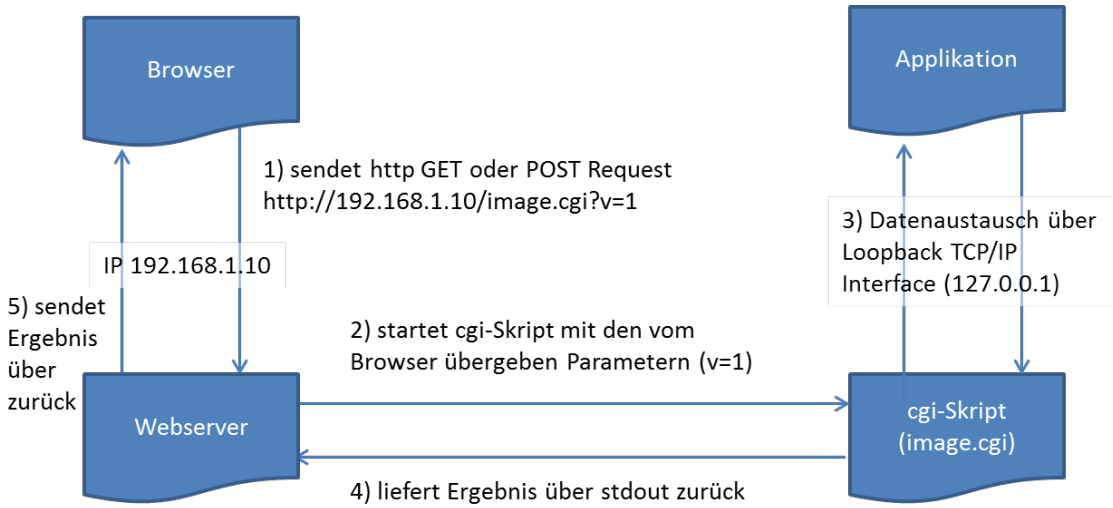


Abbildung 20: *Interprozess Kommunikation mittels cgi-Skript.*

Die Methode der cgi-Skripte zählt sicherlich nicht zu den modernsten Webtechnologien, aber ist immer noch ein beliebtes – da sehr schlankes – Mittel, um eine Applikation mit dem Webserver zu synchronisieren.

[37] Der Request ist also „stateless“ d.h. das cgi-Skript hat in der Regel bei jedem Start keine Information aus vorangegangenen Requests.
[38] Dies können auch Binärdaten in Form eines Bildes sein.
[39] Während der Ausführung des cgi-Skriptes müssen sowohl Webserver als auch Browser auf das Ergebnis warten.

Wir werden nun in der folgenden Übung das sog. Applikation Template für Entwicklungen auf der leanXcam kennenlernen, welches sich die o.g. Synchronisation zunutze macht. Allerdings sind die eigentlichen Browser Requests in Java Skript Code eingebettet, so dass wir nicht alle Details studieren können.

4.5.1. Übung 7: IPC am Beispiel des Applikation Template

Wir werden an diesem Beispiel noch 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 sehr 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 können ein Makefile aufrufen und auf Basis der Tragets ein Projekt bearbeiten.

2. Wir können ein existierendes Projekt in Eclipse importieren, die Files bearbeiten und die verschiedenen Targets des Makefiles ausführen.

3. Wir kennen die wesentlichen Komponenten des leanXcam Application Templates und wissen, an welcher Stelle individuelle Bildverarbeitungsroutinen eingefügt werden können.

4. Wir können die Komponenten IPC im Application Template lokalisieren und verstehen den grundsätzliche Aufbau.

Übung im Detail:

1. Erstellen Sie ein neues Verzeichnis `uebung07` unter `/home/oscar/uebung` und wechseln Sie in das Verzeichnis.

2. Unter `/home/oscar/uebung` das File `uebung07.tar`. Legen Sie dies im o.g. Verzeichnis ab. Entpacken Sie das tar-Archiv. Es wird ein neues Verzeichnis angelegt (`app-template`). Wechseln Sie in dieses Verzeichnis.

3. 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.

4. Studieren Sie die Ausgabe von `make` auf der Kommandozeile und versuchen Sie, in groben Zügen den `make` Prozess nachzuvollziehen.

```
oscar@oscar:~/uebung/uebung07/app-template$ make
gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_HOST -O2 -MD debug.c -o build/debug_host.o
gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_HOST -O2 -MD ipc.c -o build/ipc_host.o
gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_HOST -O2 -MD main.c -o build/main_host.o
gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_HOST -O2 -MD mainstate.c -o build/mainstate_host.o
gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_HOST -O2 -MD process_frame.c -o build/process_frame
gcc -fPIC -o app_host build/debug_host.o build/ipc_host.o build/main_host.o build/mainstate_host.
gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_HOST -O2 -MD cgi/cgi.c -o build/cgi/cgi_host.o
gcc -fPIC -o cgi/cgi_host build/cgi/cgi_host.o oscar/library/libosc_host.a -lm
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -O2 -MD debug.c -o build/debug_
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -O2 -MD ipc.c -o build/ipc_tar
```

5. Studieren Sie das Makefile und finden Sie die Targets `all:`, `clean:`, `deploy:` und `run`: Identifizieren Sie Kommandozeilen Operationen wie `ssh`, `tar`, `rm`, `cp` , die Sie bereits kennen, in den Rules zu diesen Targets.

6. Führen Sie die Targets `deploy` und dann `run` aus.

   ```
   make deploy
   make run
   ```

   Öffnen Sie den Browser und verbinden Sie sich auf die leanXcam[40].
   a. Überlegen Sie, welche Operation bei „Algorithmus anwenden“ auf dem Bild stattfindet.
   b. Bewegen Sie schnell Ihre Hand oder ein anderes Objekt vor der Kamera. Sehen Sie einen „Rolling Shutter“ Effekt? Was schliessen Sie daraus?

7. Beenden Sie `make run` mittels Ctrl C. Läuft das Programm noch auf der leanXcam? Finden Sie im Makefile das Target `run:` Die zugehörige Rule lautet[41]:

   ```
   ssh root@$(CONFIG_TARGET_IP) /mnt/app/$(APP_NAME).app/run.sh
   ```

   Versuchen Sie, die Rule zu verstehen:
   Eine SSH Verbindung als User `root` zum Host mit der IP `CONFIG_TARGET_IP`[42] wird geöffnet und im Verzeichnis `/mnt/app/$(APP_NAME)`[43] die Applikation `run.sh` gestartet.
   Studieren Sie den Inhalt des Skriptes `run.sh` (im Verzeichnis `./app` des Quellcode Verzeichnisses enthalten) und suchen Sie das Statement[44]:

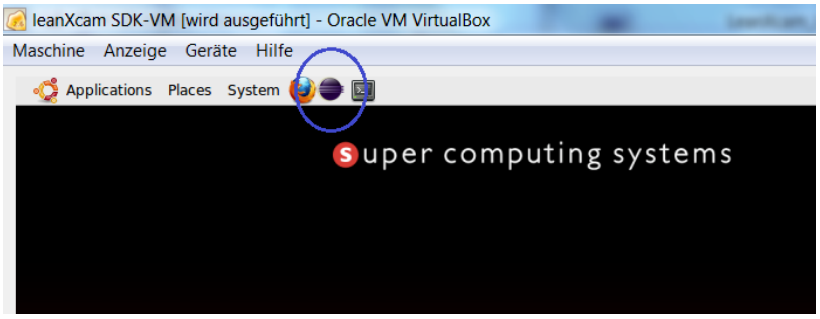
   ```
   killall app
   ```

   Es sorgt dafür, dass etwaige laufende `./app` Prozesse beendet werden. Suchen Sie das Skript `run.sh` auf der leanXcan und starten es dort direkt (im Gegensatz zu `make run` vom Host). Verifizieren Sie die Funktionalität des Programmes und beenden Sie den Prozess mit Crlt C.

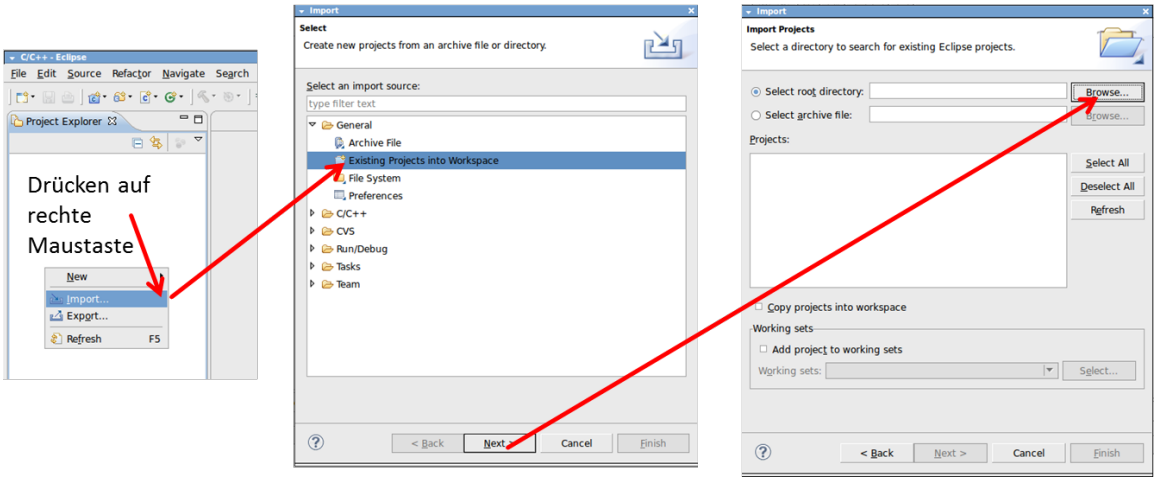
[40] Es kann vorkommen, dass das Livebild erst nach dem zweiten Ausführen des Target `run` erscheint.
[41] Die Rule hat am Ende noch das Statement `|| true` . Dieses sorgt lediglich – falls der erste Teil der Regel fehlschlägt und einem Return Wert von `false` liefert – für eine Return Wert `true`.
[42] Der Wert dieser Variablen ist im File `.config` (Ergebnis des `./configure` Skriptes) definiert. Wie findet des Makefile dieses File? (mittels `-include .config`)
[43] Der Wert dieser Variablen ist im oberen Teil des Makefiles definiert.
[44] Der Teil `2> /dev/null` unterdrückt nur die Error Ausgabe.

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.

8. Öffnen Sie das Eclipse IDE[45] durch Drücken des entsprechenden Buttons auf dem VM Desktop.



9. Importieren Sie das Projekt app-template:

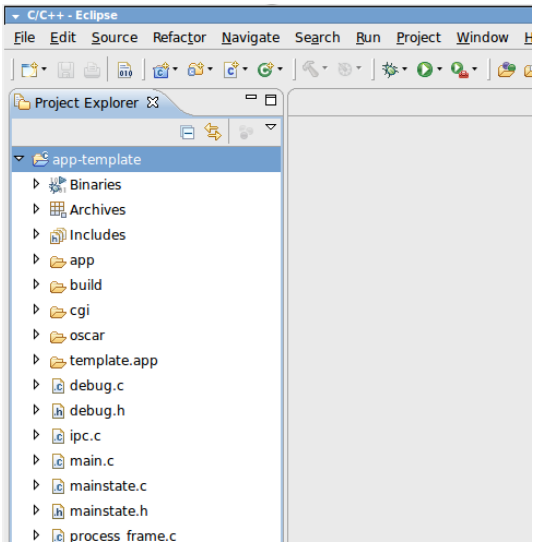


Rechts, bei `Browse`, das Verzeichnis

`~/uebung/uebung07/app-template`

auswählen und `Finish` drücken. Dann ist das gesamte Projekt im Filetree links vorhanden:

[45] Integrated Development Environment

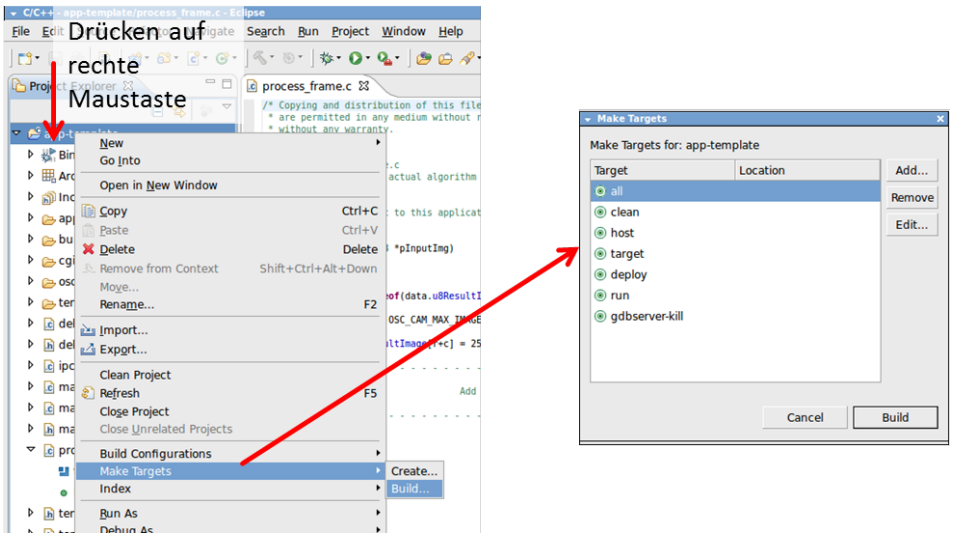


10. Machen Sie sich ein wenig mit dem Eclipse Tool vertraut, indem Sie durch den Source Code Browsen. Identifizieren Sie im File

    `process_frame.c`

    die Stelle, an der die Verarbeitung für „Algorithmus anwenden“ stattfindet (Inversion des Bildes durch: `Grauwert = 255 - Grauwert`) Hier werden wir in der Zukunft unsern eigenen Code „einbauen“.

11. Verwenden Sie nun die Eclipse Umgebung, um die Targets zu erzeugen. Gehen Sie wie folgt vor:



    Wählen Sie die entsprechenden Targets aus, um das Programm zu erstellen und zu „deployen“. Verwenden Sie das Target `run` nur von der Kommandozeile, ausser Sie wissen, wie Sie innerhalb von Eclipse ein Ctrl-C an die Konsole schicken können.

12. Lokalisieren Sie das cgi-Skript im Projekt (Quellcode):

```
./cgi/cgi.c
./cgi/cgi.h
```

und auf der leanXcam HW (Binärcode):

```
/home/httpd/cgi-bin/bin
```

Suchen Sie das Statement

(cgi.c, Zeile 157) `case APP_CAPTURE_ON`

an und analysieren Sie, in welcher Form das geforderte Bild an den Webserver übermittelt wird (Hinweis: `return OscBmpWrite(&pic, IMG_FN);`).
Finden Sie nun das Bild auf dem Target

```
/home/httpd/image.bmp
```

13. Das File `ipc.c` ist das Gegenstück zu `cgi.c` in der Applikation, d.h. ist dort für die IPC verantwortlich. Versuchen Sie, ausgehend von den Aufrufen an das OSCar-Framework

    `CheckIpcRequests (..)` (in `./ipc.c`)
    `HandleIpcRequests (..)` (in `./ mainstate.c`)

    die Implementierung der IPC im Application Template grob nachzuvollziehen.

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_frame.c` zu integrieren) und per Browser in Echtzeit anzuzeigen.
- 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 OSCar Frameworks ein Tool an der Hand, um einfach auf die HW-Ressourcen der leanXcam zugreifen zu können, um unsere Anwendungen sukzessive zu verfeinern.

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[46] abgelegt, welches bei der Ausführung von `make` gelesen wird. Die wichtigsten Targets im Makefile des Application Templates (s. Übung 7, 4.5.1) sind:

- `all:`
  Default Target, welches das gesamte Projekt erstellt.

- `clean:`
  Löscht alle Objekt- und Binärfiles.

- `deploy`:
  Kopiert den Binärcode per SSH-Verbindung auf die leanXcam (es muss das Passwort `oscar` eingegeben werden).

- `run`:
  Führt den Binärcode per SSH-Verbindung auf dem leanXcam Target aus (es muss das Passwort `oscar` eingegeben werden).

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
```

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

```
make
```

5.2. Erweiterte Einführung in Makefiles

[47]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

[46] Makefile ist hier als Platzhalter für irgendein File mit Rules für `make` verwendet. Meist wird aber der Name auch direkt als Filename verwendet.

[47] Die folgenden Seiten wurden im Wesentlich „as is“ von [12] übernommen, um einige Grundlagen zu Makefiles zum direkten Nachschlagen zur Verfügung zu haben.

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.

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 Objektdateien 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):

$<	die erste Abhängigkeit
$@	Name des targets
$+	eine Liste aller Abhängigkeiten
$^	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 sollen verschiedene Anwendungen der leanXcam in Form von Beispielprogrammen erarbeitet werden.

6.1. Übung 8: Textur-basierte Change Detection

Wir werden in diesem Beispiel die Testaufgabe 1 auf der leanXcam implementieren. Hierzu werden wir ausgehend von der Prototypversion unter Matlab eine vereinfachte Variante in C programmieren. Dabei starten wir mit einem Gradienten Filter aus Kapitel 3.1.2 des Theorie Skriptes [13]. Zusätzlich wollen wir im Rahmen dieser Übung den Umgang mit dem Versionierungstool `git` [8] kennenlernen.

Ziele:

1. Wir können Änderungen an unseren Programmen mittels des Versionierungstools `git` tracken und jederzeit zu einer älteren Version zurückkehren.

2. Wir können Gradienten Filter auf der leanXcam verwenden.

3. Wir verstehen die Ressourcenproblematik der Implementierung auf einem DSP.

4. Wir kennen die Grundlagen der Change Detektion.

Übung im Detail:

1. Wechseln Sie in das Verzeichnis:

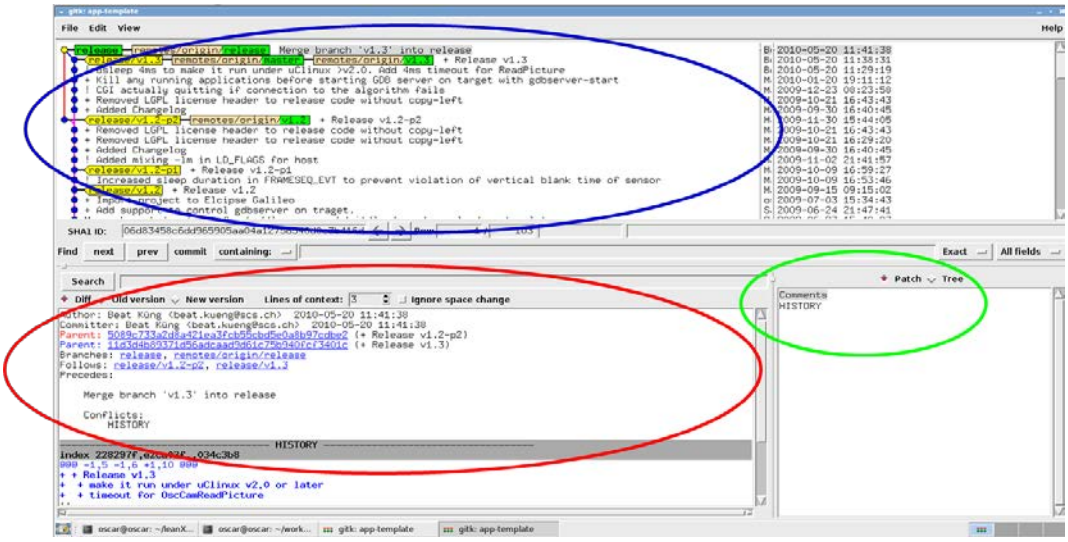
   ```
   /home/oscar/leanXcamSDK/app-template
   ```

   Es enthält die ursprüngliche Version des Applikation Templates, welches vom Hersteller der leanXcam via ein git Repository zur Verfügung gestellt wird[48] (s. auch Abschnitt 4.2.1). 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 Repository `app-template.git`.

[48] Der Zugriff auf das Repository erfolgt mittels: `git clone git://github.com/scs/app-template.git`



In dem Fester sehen Sie:

a. Die verschiedenen Versionen (blau)
b. Wenn Sie eine der Versionen anwählen, die Liste der für diese Version geänderten Files (grün)
c. Wenn Sie eines der Files anwählen, die Änderungen im File.

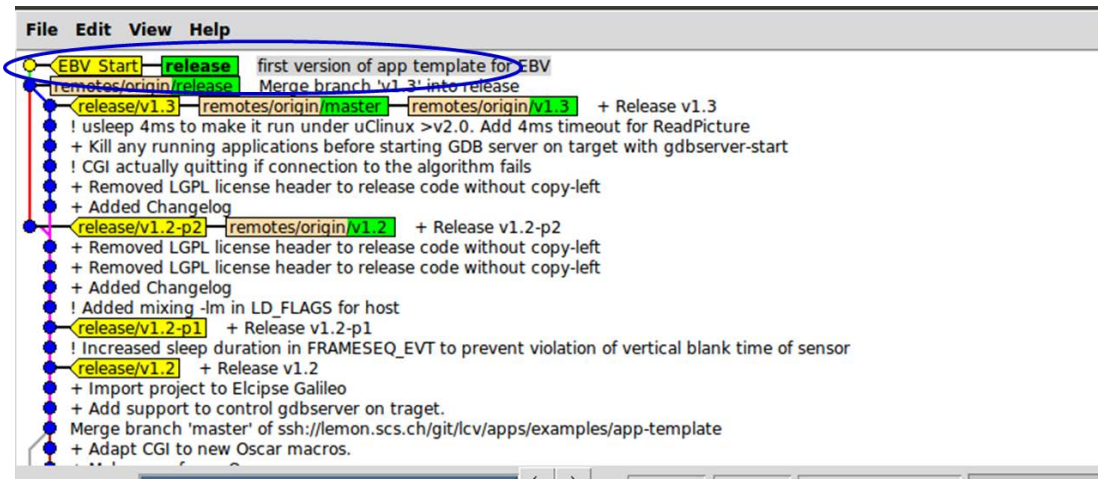
2. Wir wollen nun im Folgenden dieses Repository verwenden, um die Code Änderungen und Applikationsentwicklungen zu verfolgen. Hierzu werden wir zuerst die Version des Applikation Templates aus Übung 7 in Kapitel 4.5.1 aus einem Remote Repository in unser lokales Repository übertragen. Dies geschieht mit folgendem `git` Befehlt:

   ```
   git pull -t git://github.com/klaus-zahn/app-template.git master
   ```

   Wenn Sie nun wieder mittels

   ```
   gitk –-all
   ```

   die Versionshistorie untersuchen, so stellen Sie fest, dass ein neuer Eintrag hinzugekommen ist (blau)::



   Sie sehen nun an oberster Stelle in der Versionsliste den Tag `EBV_Start`,

welcher mit der Version aus Übung 7 identisch ist. Der gelbe Punkt vor dem Tag `EBV_Start` zeigt Ihnen, dass Sie aktuell diese Version auch ausgecheckt haben. Von dieser Version ausgehend wollen wir nun die Texturbasierte Change Detektion implementieren.

3. Da das vorliegende Projekt bisher noch nie erstellt wurde, muss es zuerst konfiguriert werden. Dies erfolgt mit dem Befehl:

   `./configure`

```
oscar@oscar:~/leanXcamSDK/app-template$ ./configure
Is the Oscar Framework already in the folder ./oscar? (y/n) [n]: n
Enter the path to the Oscar Framework. [../oscar]: ../oscar
Enter the IP Address of the target device. [192.168.1.10]:
Do you want to enable debugging symbols? (y/n) [n]: n
Do you want to enable IO simulation on the target? (y/n) [n]: n
Select the board you are using (Mesa-SR4k/leanXradio/leanXcam/indXcam). [leanXcam]:
make[3]: Nothing to be done for `reconfigure'
```

   Wesentlich dabei ist vor allem, den Link zum OSCar Framework herzustellen. Testen Sie dann, ob das Projekt erstellt werden kann und auf dem Target läuft.

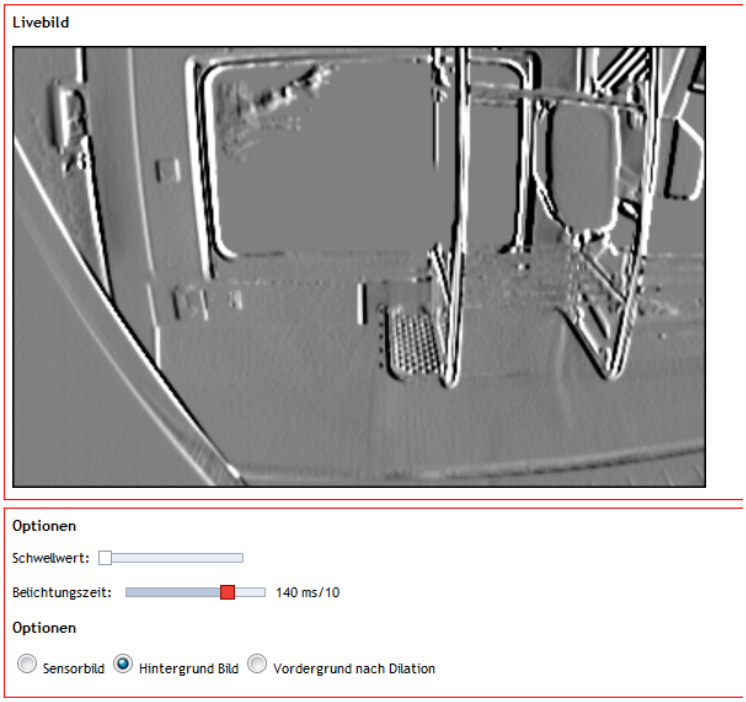
4. Ändern Sie nun die Applikation ab, indem Sie in der Methode `ProcessFrame` einen Sobel Kantenfilter implementieren und diesen auf dem Web GUI anzeigen lassen.
   Die Implementierung eines Kantenfilters in C ist relativ „straigth foreward“:

```
int c, r;
int nc = OSC_CAM_MAX_IMAGE_WIDTH/2;/* we work on half of the camera image width */
int siz = sizeof(data.u8TempImage[GRAYSCALE]);

for(r = nc; r < siz-nc; r+= nc)/* we skip the first and last line */
{
    for(c = 1; c < nc-1; c++)
    {    /* do pointer arithmetics with respect to center pixel location */
        unsigned char* p = &data.u8TempImage[GRAYSCALE][r+c];
        /* implement Sobel filter (shift by 128 to 'center' grey values */
        short v = 128  -(short) *(p-nc-1) + (short) *(p-nc+1)
                  -2* (short) *(p-1) + 2* (short) *(p+1)
                     -(short) *(p+nc-1) + (short) *(p+nc+1);

        /* apply min()/max() to avoid wrap around of values below 0 and above 255 */
        data.u8TempImage[BACKGROUND][r+c] = (unsigned char) MAX(0, MIN(255, v));
    }
}
```

   Der obige Code realisiert einen Sobel-Kantenfilter in x-Richtung (`nc` ist die Länge einer Bildzeile, `siz` die Grösse des gesamten Bildes). Dabei sind die Filterkoeffizienten gelb markiert (ein Vorzeichen allein steht für +1 bzw. -1). Die Anwendung des `MIN()` bzw. `MIN()` Operators ist notwendig, um die Werte wieder auf das Intervall `[0,255]` zu beschränken.
   Realisieren Sie ein Kantenfilter in x- und in y-Richtung (Koeffizienten anpassen!), wobei Sie die Ergebnisse einfach in die Bildspeicher `[BACKGROUND]` (für x-Richtung wie oben) bzw. `[THRESHOLD]` (für y-Richtung) schreiben. Somit können Sie beide Ergebnisse auch direkt im Web GUI der Applikation visualisieren:



Bemerkungen:
a) Der additive Term 128 in der Berechnung ist nur für die Anzeige gedacht und sollte für die folgende Implementierung der Change Detektion gelöscht werden.

b) Für eine Anwendung, bei der die Kantenbilder angezeigt werden sollten, sollten Sie aus „kosmetischen“ Gründen noch die Beschriftung der beiden Bilder im Web GUI anpassen (am Radio Button). Dies würde im File `cgi/www/index.xhtml` (Zeilen 97 und 101) erfolgen.

5. Damit können wir nun zur Implementierung der Change Detektion schreiten, welche ja im Wesentlichen auf der Berechnung der Ableitung beruht. Hierbei benötigen wir noch verschiedene Hilfsvariablen:

   a) Schrittzähler, welcher für die Initialisierung von Werten beim Start
   `data.ipc.state.nStepCounter == 1`
   verwendet werden kann.

   b) Schwellwert, für die Einstellung der Sensitivität. Dieser kann live im Web GUI mit dem entsprechenden Slider angepasst werden.
   `data.ipc.state.nThreshold`

   c) zusätzliche Bildspeicher, welche im File template.h (ab Zeile 60) im enum `IMG_TYPE` hinter `THRESHOLD` hinzugefügt werden können.

```
enum IMG_TYPE
{
    GRAYSCALE,
    BACKGROUND,
    THRESHOLD,
    MAX_NUM_IMG
};
```

Bemerkung:
Beachten Sie, dass die Bilder in den Speichern korrespondierend zu den Indizes `BACKGROUND` und `THRESHOLD` angezeigt werden.

6. Implementieren Sie nun eine *vereinfachte* Version der Texturbasierten Hintergrundschätzung. Dies vor allem auch, weil der DSP keine FPU (Floating Point Unit) besitzt:

   a) Nähern Sie die Norm der Ableitung (s. Skript Abschnitt 3.1.2)

   $$\frac{dI}{dr} = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

   durch die Summe der Beträge an (machen Sie sich klar, worin der Fehler besteht):

   $$\frac{dI}{dr} \approx \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right|$$

   Dies erspart die Berechnung der Quadrate und vor allem der Quadratwurzel.

   b) Wir implementieren die Hintergrundschätzung – um Ressourcen zu sparen – allerdings nur mit zwei Winkel Bins. D.h. wir unterscheiden nur zwischen Werten der Ableitung $\left|\frac{\partial I}{\partial x}\right| > \left|\frac{\partial I}{\partial y}\right|$ bzw. $\left|\frac{\partial I}{\partial x}\right| \leq \left|\frac{\partial I}{\partial y}\right|$ und natürlich der Frage der Norm (bzw. unserer Näherung) in Bezug auf den Schwellwertes. Dies bedeutet dann, dass der Pixelzähler $\texttt{Bkgr}_{ij}\texttt{(n)}$ nur für drei Werte `n = 0,1,2` benötigt wird:

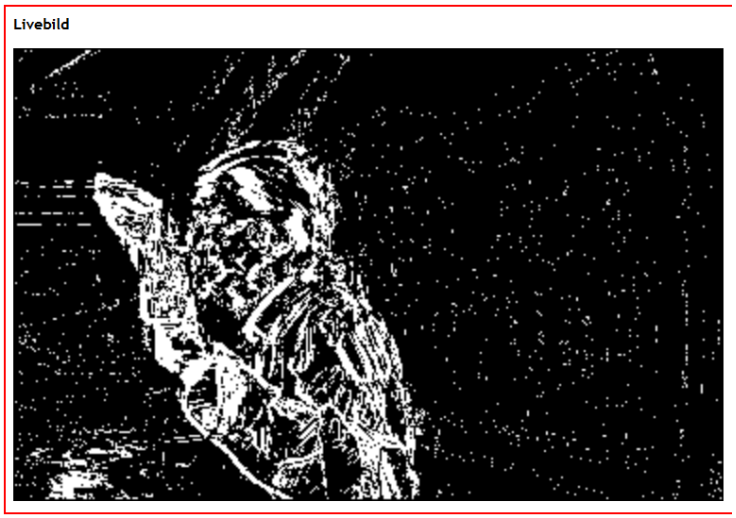
   $$n = 0 \quad \Leftrightarrow \quad \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right| < \text{Threshold}$$

   $$n = 1 \quad \Leftrightarrow \quad \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right| \geq \text{Threshold} \quad \wedge \quad \left|\frac{\partial I}{\partial x}\right| > \left|\frac{\partial I}{\partial y}\right|$$

   $$n = 2 \quad \Leftrightarrow \quad \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right| \geq \text{Threshold} \quad \wedge \quad \left|\frac{\partial I}{\partial x}\right| \leq \left|\frac{\partial I}{\partial y}\right|$$

   Implementieren Sie diese Lösung und observieren Sie das Detektionsverhalten als Funktion der Parameterwerte (Schwellwert bzw. Belichtungszeit). Welche Framerate erhalten Sie? Wo sehen Sie noch Optimierungspotential.

   Das Ergebnis des Vordergrundbildes sollte dem folgenden Bild ähnlich sehen:



7. Nach dem Beenden der Implementierung wollen wir die Änderungen in das Repository übertragen (sog. „commit“). Hierzu geben Sie auf der Kommandozeile folgenden Befehl ein:

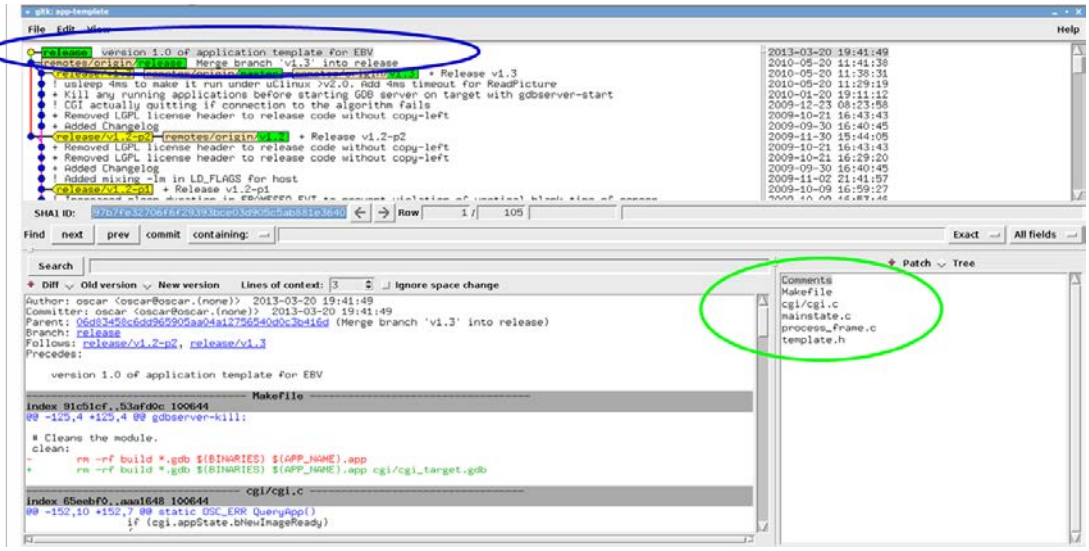
   ```
   git commit -a -m "version 1.0 of texture based change detection"
   ```

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

8. Wenn Sie nun wieder mittels

   ```
   gitk --all
   ```

   die Versionshistorie untersuchen, so stellen Sie fest, dass ein neuer Eintrag hinzugekommen ist (blau):



   Sie können noch verifizieren, ob die Liste der geänderten Files (grün) stimmt.

6.2. Übung 9: Change Detection mit Morphologie und Region Labeling

Wir werden in diesem Beispiel die Übung 8 mit morphologischen Operationen und dem Region Labeling erweitern.

Ziele:

1. Wir können einfache morphologische Operationen in C auf der leanXcam implementieren.

2. Wir können ein Region Labeling mit Hilfe des OSCar Frameworks durchführen.

Übung im Detail:

1. Ziehen Sie sich vom gitbhub Server den Branch „ubeung09_start“:

   ```
   git pull -t git://github.com/klaus-zahn/app-template.git uebung09_start
   ```

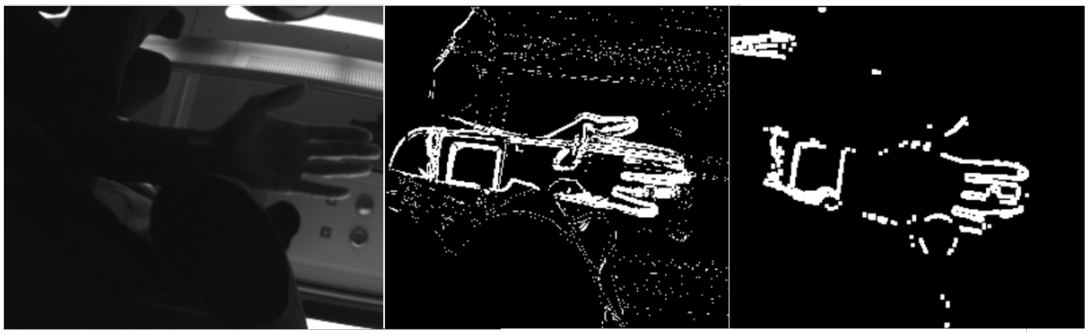
2. Studieren Sie das File `process_frame.c`. Es enthält die Lösung von Übung 8 (Kapitel 6.1), wobei der Programmcode etwas umstrukturiert wurde. Der Bildspeicher zum Index `THRESHOLD` enthält nun das Vordergrundbild. Wir wollen nun in einem ersten Schritt eine morphologische Öffnung (Opening) und dann eine Schliessung (Closure) implementieren.

3. Implementieren Sie zuerst eine Öffnung auf dem Bildspeicher `THRESHOLD` d.h. zuerst eine Erosion dann eine Dilation. Ziel der Operation ist es, isolierte Vordergrundpixel, welche durch Rauschen im Bild entstehen, zu eliminieren. Dabei ist – ähnlich wie bei der Implementierung des Kantenfilters – eine Implementierung einer Erosion bzw. Dilation recht einfach in C. Wir beschränken uns dabei auf ein Strukturelement bestehend aus einer 3x3 Einheitsmatrix. Eine Erweiterung ist „straight forward“:

   ```
   for(r = nc; r < siz-nc; r+= nc)/* we skip the first and last line */
   {
       for(c = 1; c < nc-1; c++)
       {
           unsigned char* p = data.u8TempImage[INPUT][r+c];
           data.u8TempImage[OUTPUT][r+c] = *(p-nc-1) & *(p-nc)  & *(p-nc+1) &
                                           *(p-1)    & *p       & *(p+1)    &
                                           *(p+nc-1) & *(p+nc) & *(p+nc+1);
       }
   }
   ```

   Durch die logische `UND` (&) Operation der Pixel wird eine Erosion realisiert (nc ist die Länge einer Bildzeile), während eine logische `ODER` (|) Operation der Pixel eine Dilation erzeugen würde.
   Die Wirkung der Öffnung auf eine reale Aufnahme ist im folgenden Bild dargestellt. Das Sensorbild (links) nach der Schwellwertoperation mit dem Hintergrundbild (mitte) und nach der Öffnung (rechts)[49].

[49] Die Bilder wurden mit der leanXcam aufgenommen und korrespondieren daher nicht genau.



Wählen Sie zur Übersichtlichkeit für die Implementierung der Öffnung eine eigene Funktion nach der Vorlage im Ausgangscode.

4. Fügen Sie nun noch eine Schliessung (Closure) d.h. ein Dilatation mit nachfolgender Erosion an. Ziel dieser Operation ist es, das Zerfallen von grösseren Objekten in kleinere, welches durch die Öffnung favorisiert wurde, zu vermeiden. Die Wirkung der Schliessung auf eine reale Aufnahme ist im folgenden Bild dargestellt. Das Sensorbild (links) nach der Öffnung (mitte) und nach der Schliessung (rechts)[49].

5. Nun implementieren wir das Region Lableing, wobei wir uns hierfür auf die Verwendung einer im OSCar Framework bereits existierenden Funktion beschränken:

```
OscVisLabelBinary(struct OSC_PICTURE *picIn, struct OSC_VIS_REGIONS
                                                        *regions)
```

Hierzu müssen Sie das Input Bild (`picIn`) in die Struktur `OSC_PICTURE` wrappen. Beachten Sie, dass die Vordergrund Pixel des Input Bildes (`picIn`) den Wert `0x01` (und nicht `0xff`) haben müssen. Dies können Sie entweder durch Anpassung der Erstellung des `THRESHOLD` Bildes oder durch Konversion des `THRESHOLD` Bildes mit Hilfe der Funktion `OscVisGrey2BW` erreichen. Weiterhin müssen Sie eine Struktur `OSC_VIS_REGIONS` deklarieren und beim Aufruf an die Funktion `OscVisLabelBinary` übergeben.

6. Ebenso gibt es im OSCar Framework eine Funktion für die Extraktion der ROI Eigenschaften (d.h. Bounding Box und Massenmittelpunkt).

```
OscVisGetRegionProperties(struct OSC_VIS_REGIONS *regions)
```

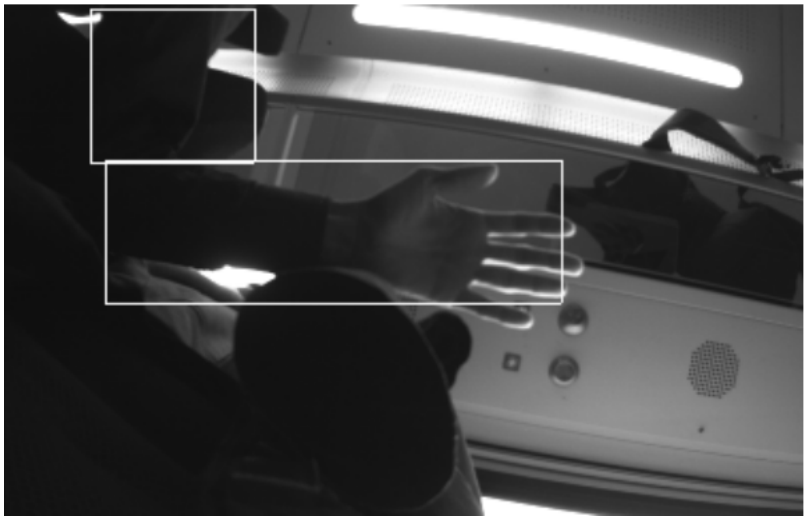
Hierbei müssen Sie wieder die Struktur `OSC_VIS_REGIONS` vom Aufruf an die

Funktion OscVisLabelBinary übergeben.

7. Zur Verifizierung des Ergebnisses sollten Sie die Bounding Box in ein Ausgabebild einzeichnen. Es gibt im OSCar Framework eine Routine (`OscVisDrawBoundingBox`) hierfür, die allerdings nur für Farbbilder funktioniert. Daher wurde eine Version für Grauwertbilder im Ausgangscode implementiert[50].

```
OscVisDrawBoundingBoxBW(struct OSC_PICTURE *picIn, struct
                    OSC_VIS_REGIONS *regions, uint8 Color, int MinSize)
```

Das Ergebnis bei Einzeichnen in das Grauwertbild sollte etwas so aussehen (Bounding Box mit Argument `Color 0xFF = 255`).



[50] Dabei darf die Bounding Box natürlich nur für Testzwecke in das Grauwertbild eigezeichnet werden, weil sonst ja die Bildinformation geändert wird.

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 21). Eine allgemeine Einführung zu den Netzwerkeinstellungen von VirtualBox findet sich in [14]. Ein grundsätzliches Verständnis der Wirkungsweise von NAT, Netzwerkbrücke und Host-only Adapter (unter *Angeschlossen an*) sollte vorhanden sein.

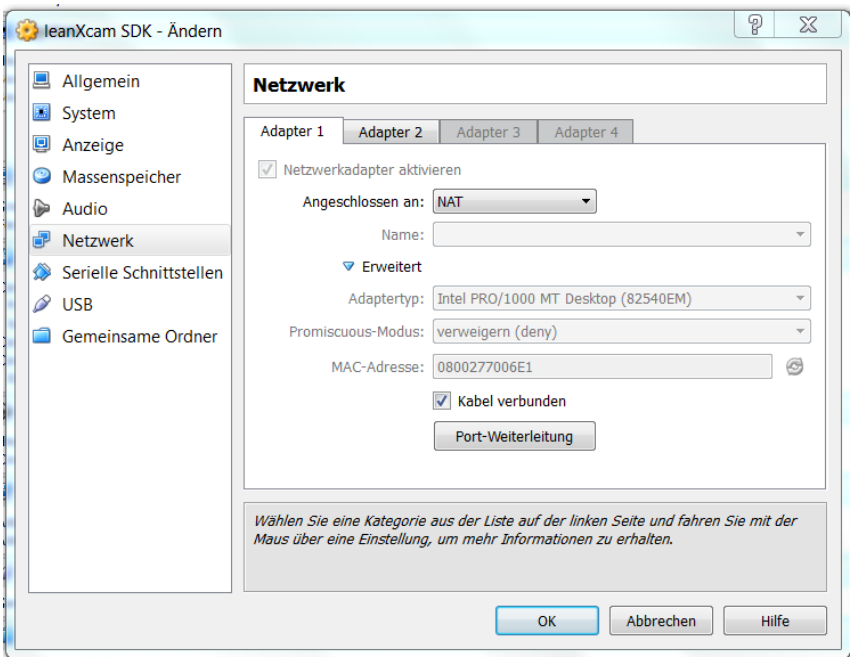
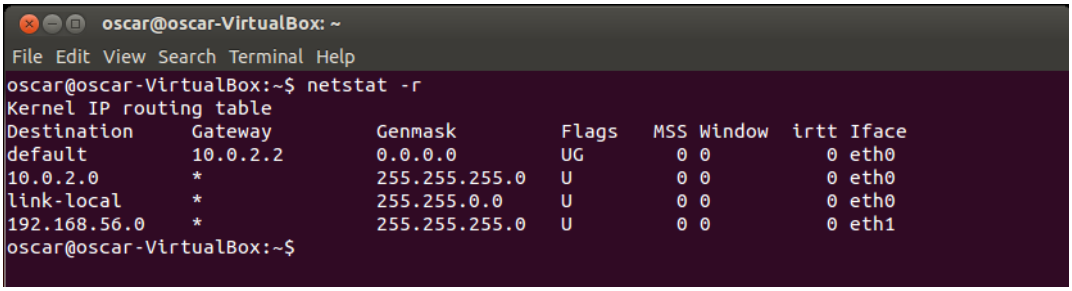


Abbildung 21: *Dialog zur Einstellung der Netzwerk Settings.*

Um sowohl eine Verbindung zum Internet (und zur leanXcam, 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[51].
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 22).
4. In der VM ggf. das Netzwerk neu starten: *sudo /etc/init.d/networking restart*
5. Falls es noch Probleme mit dem Internetzugriff gibt, die Routing Tabelle überprüfen: *netstat –r*

[51] 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"*



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 eth6. Vorher ggf. die *default* Route mit *sudo route del default* löschen.

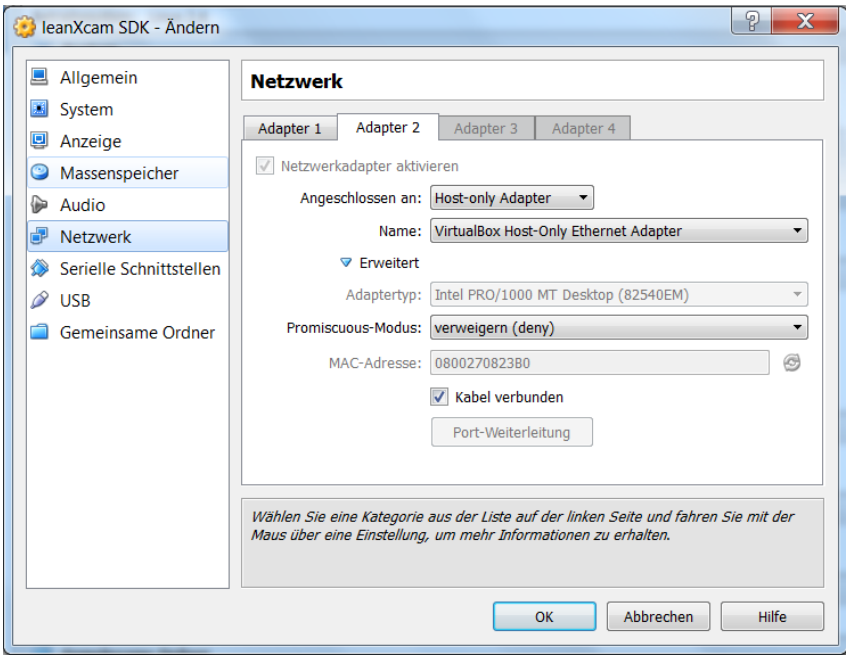


Abbildung 22: *Einstellungen für Host-only Adapter.*

8. leanXcam

8.1. Das Wichtigste in Kürze

Die wichtigsten relevante Befehle und Informationen zur Verwaltung der leanXcam finden sich unter [15].

Persistente Speicherung von Files:

- Nur der Inhalt des Verzeichnisses `/mnt/app` ist persistent im Flash gespeichert. Alle anderen Verzeichnisse werden beim Reboot aus dem Boot Flash erneuert. Daher sollte die Applikation im Verzeichnis `/mnt/app` abgelegt werden.
- Allerdings ist das Verzeichnis `/mnt/app` eher langsam und auf 4 MB begrenzt.

`fw_printenv`
Zeigt die Environment Variablen des U-Boot Loaders an. Unter anderem stehen hier die Werte der

- IP Adresse und Netzmaske
- MAC Adresse
- das `runscript`, also die Applikation, die nach dem Bootvorgang ausgeführt wrid

```
root:/mnt/app/web_view_cpp.app> fw_printenv
bootargs=root=/dev/mtdblock0 rw console=ttyBF0,115200
bootcmd=run boot
bootdelay=2
baudrate=115200
loads_echo=1
autoload=no
rootpath=/romfs
netmask=255.255.255.0
hostname=bf537-leanXcam
loadaddr=0x1000000
flash_base_linux=0x10028000
flash_base_monitor=0x10000000
loadaddr-linux=0x2000000
boot=bootm $(flash_base_linux)
nokgdbargs=setenv bootargs root=/dev/mtdblock0 rw console=ttyBF0,115200
nfsargs=setenv bootargs root=/dev/nfs rw
nfsroot=$(serverip):$(rootpath) console=ttyBF0,57600
upduboot=tftp $(loadaddr) u-boot.ldr; cp.b $(loadaddr) $(flash_base_monitor) $(filesize)
updlinux=tftp $(loadaddr-linux) uImage; cp.b $(loadaddr-linux) $(flash_base_linux) $(filesize)
tstuboot=tftp $(loadaddr) u-boot.bin; go $(loadaddr)
tstlinux=tftp $(loadaddr-linux) uImage; bootm $(loadaddr-linux)
ipaddr=192.168.1.10
serverip=192.168.1.2
gatewayip=192.168.1.2
runscript=web_view_cpp.app/run.sh
ethaddr=00:a0:10:00:b1:b5
hwrev=LX_1.1_B
root:/mnt/app/web_view_cpp.app>
```

`fw_setenv runscript web_view_cpp.app/run.sh`
Setzt den Wert der Variablen `runscript` auf `web_view_cpp.app/run.sh`

`fw_setenv ipaddr 192.168.1.10`
Setzt den Wert der Variablen `ipaddr` auf `192.168.1.10`

8.2. Logging

Im Code finden sich zahlreiche Log-Ausgaben, die sowohl auf die Kommandozeile als auch in ein File geschrieben werden. Z.B. mit dem Kommando

```
OscLog(ERROR, "%s: IPC request error! (%d)\n", __func__, err);
```

wird mit dem Log-Level `ERROR` die Nachricht, welche als zweites Argument im `OscLog` Befehl auftritt, ggf. auf die Kommandozeile und ins File geschrieben. Die Komposition der Nachricht folgt der C-Format String Konvention: So werden `%s` und `%d` für einen String bzw. einen Integer Wert verwendet, welche – in der korrekten Reihenfolge – in der Argumenten Liste des `OscLog` Befehl ab der dritten Position auftreten (`__func__, err`)[52].
Ob ein Wert wirklich auf die Kommandozeile bzw. das Logfile geschrieben wird, hängt davon ab, welches maximale Log-Level für die beiden möglichen Ausgabepfade gesetzt wurden. Dies erfolgt (zum Programmstart) mittels:

```
OscLogSetConsoleLogLevel(INFO);
OscLogSetFileLogLevel(WARN);
```

Hier wird für die Konsole das maximale Log-Level auf `INFO` gesetzt und für das Log File auf `WARN`. Die Hierarchie der Log-Level ist in der folgenden Abbildung 23 zu sehen (OSCar Framework: **log.h** [code]):

```
enum EnOscLogLevel {
        NOLOG = -1, /* not allowed for OscLog() */
        EMERG,
        ALERT,
        CRITICAL,
        ERROR,
        WARN,
        NOTICE,
        INFO,
        DEBUG,
        NONE,
        SIMULATION = 255
};
```

Abbildung 23: *Die verschiedenen Log-Level.*

So werden z.B. auf der Konsole alle Log-Level „oberhalb“ von `INFO` ausgegeben während die Einstellung für das Log File etwas restriktiver ist.

Das Log File der Applikation findet sich unter:

```
/var/log/log
```

Ein praktischer Befehl, um die Änderungen im Log File live mit zu verfolgen ist:

```
tail –f /var/log/log
```

Hiermit wird das Ende des Files angezeigt und aufgrund der `–f` Option („follow“) auch neu hinzukommende Einträge live ausgegeben.

[52] Die Präprozessor Direktive `__func__` wird bei der Erstellung des Codes durch den Funktionsnamen ersetzt.

8.3. Debugging auf dem Target

Hartnäckige Fehler lassen sich in der Regel nur durch zeilenweises Debugging des Source Codes finden. Der Standard Debugger unter Linux ist `gdb` (GNU Debugger). Allerdings kann der Debugger aus Performance Gründen (ebenso wie der Compiler) nicht auf dem Target (leanXcam) laufen. Daher muss der Debugger mittels einer Remote Verbindung gesteuert werden.
Hierzu ist wie folgt vorzugehen:

1. Erstellen des Codes mit Debug Informationen:
   Der Code ist per Default mit der Option `-O2` auf maximale Geschwindigkeit kompiliert. Dies ist an der Konsolenausgabe während des `make` Vorganges zu erkennen:

```
gcc -fPIC -o cgi/cgi_host build/cgi/cgi_host.o oscar/library/libosc_host.a -lm
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -O2 -MD debug.c -o build/debug_target.o
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -O2 -MD ipc.c -o build/ipc_target.o
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -O2 -MD main.c -o build/main_target.o
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -O2 -MD mainstate.c -o build/mainstate_target.o
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -O2 -MD process_frame.c -o build/process_frame_target.o
process_frame.c: In function 'ProcessFrame':
```

   Der sog. Optimizer ist für diese Optimierung verantwortlich. Er nimmt, je nach Einstellungen (`-O0, -O1, -O2, -Os`), unterschiedliche Änderungen am Code vor, welche die gewünschten Optimierungsschritte erzielen. Dadurch ändert sich die Codestruktur teilweise signifikant und ein zeilenweises Debugging ist nicht mehr möglich. Daher ist der Code für eine Debugging Session mit besonderen Flags zu erstellen[53]. Hierzu ist der Aufruf `make` folgendermassen abzuändern (ggf. ist vorher ein `make clean` aufzurufen):

   ```
   make CONFIG_ENABLE_DEBUG=y
   ```

   Die Kommandozeilenausgabe sieht dann wie folgt aus:

```
gcc -fPIC -o app_host build/debug_host.o build/ipc_host.o build/main_host.o build/mainstate_host.o build/process_frame_host.o osc
gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_HOST -g -MD cgi/cgi.c -o build/cgi/cgi_host.o
gcc -fPIC -o cgi/cgi_host build/cgi/cgi_host.o oscar/library/libosc_host_dbg.a -lm
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -ggdb3 -MD debug.c -o build/debug_target.o
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -ggdb3 -MD ipc.c -o build/ipc_target.o
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -ggdb3 -MD main.c -o build/main_target.o
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -ggdb3 -MD mainstate.c -o build/mainstate_target.o
bfin-uclinux-gcc -c -std=gnu99 -Wall -Ioscar/include -DOSC_TARGET -ggdb3 -MD process_frame.c -o build/process_frame_target.o
```

   Fall die Ausgabe nicht korrekt (bzw. wie oben) erscheint, nochmals die Syntax des `make` Befehles überprüfen.
   Danach den Code mittels

   ```
   make deploy
   ```

   auf das Target kopieren und mit

   ```
   make run
   ```

   kurz testen[54]. Dann das Programm anhalten. Nun sind wir bereit, das Programm zu debuggen.

[53] Ein besonders unangenehmer Fehlerfall tritt dann auf, wenn die nicht optimierte Version –O0 stabil läuft und nur die optimierte –O2 den Fehler aufweist.
[54] Das Aufrufen von `make run` ist auch daher wichtig, um den geänderten Code auf dem Target zu entpacken.

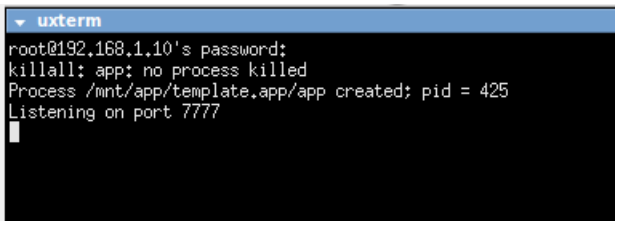
2. Starten des Servers auf dem Target:
   Wie bereits erwähnt, kann der Debugger nicht vollständig auf dem Target laufen, sondern es muss mit einer Client-Server-Lösung gearbeitet werden. Hierzu wird der Server auf dem Target gestartet. Die einfache Variante besteht in der Verwendung der Regel `gdbserver-start` des `Makefile`:

   ```
   make gdbserver-start
   ```

   Diese Regel führt auf dem Target den folgenden Befehl aus:

   ```
   /mnt/app/template.app> gdbserver :7777 ./app
   ```

   Der `gdbserver` wird gestartet zum Debuggen des Programmes[55] `./app` und „horcht“ auf dem Port 7777 auf eine Verbindung. Damit ist das Target bereit zum Debugging.



3. Starten des gdb-Debbugers auf der leanXcam VM:
   Der eigentliche Debugger wird nun auf der leanXcam VM gestartet und zwar im gleichen Verzeichnis wie die Applikation. Dies erfolgt mittels:

   ```
   bfin-uclinux-gdb app_target.gdb
   ```

   Wichtig zu beachten hierbei ist die Verwendung des Debuggers `bfin-uclinux-gdb` sowie des Binärfiles `app_target.gdb` für das Blackfin Target. Mit dem Start des Debuggers öffnet sich eine Kommandozeile, über die der Debugger gesteuert wird.

```
oscar@oscar:~/leanXcamSDK/app-template$ bfin-uclinux-gdb app_target.gdb
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=bfin-uclinux"...
(gdb)
```

4. Verbindung mit dem `gdbserver`:
   Um die Verbindung mit dem `gdbserver` aufzunehmen, muss der folgende Befehl auf der `gdb` Kommandozeile ausgeführt werden:

   ```
   target remote 192.168.1.10:7777
   ```

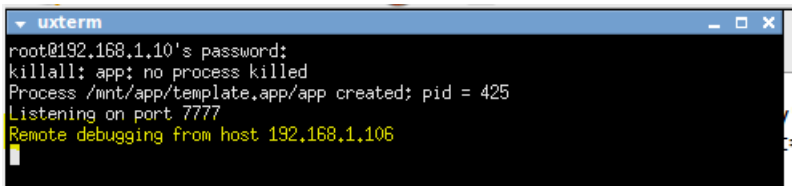
   Dabei besteht das Argument des Befehls allgemein aus der IP-Adresse (192.168.1.10) des Targets und dem Port (7777), auf welchem der `gdbserver` gestartet wurde. Auf der gdb-Kommandozeile sieht die Ausgabe

[55] Man beachte die Pfadangabe.

folgendermassen aus:

```
(gdb) target remote 192.168.1.10:7777
Remote debugging using 192.168.1.10:7777
0x00800044 in _stext ()
(gdb)
```

Auf dem Target kann ebenfalls die Verbindungsaufnahme nachvollzogen werden:



Ab diesem Punkt verhält sich der Debugger genauso, als ob ein Target lokal untersucht würde.

5. Einige wichtige Befehle des GNU Debuggers[56]:
   Ähnlich wie bei der ersten Verwendung der Linux Kommandozeile ist auch die Verwendung des GNU Debuggers gewöhnungsbedürftig. Trotzdem kann, nach einiger Übung, ein Programm sehr einfache untersucht werden.

   a. `run args`: starte das Programm mit den Kommandozeilen Argumenten `args`

   b. `break main.c:35`: Setze eine Breakpunkt im File main.c in Zeile 35 bzw.:

      `break ProcessFrame:` Setze einen Breakpunkt in der Methode `ProcessFrame`

   c. `p wert`: Geben den Werte der Variablen `wert` aus; hierbei können Notationen wie Pointer `wert->a` oder `*wert` direkt verwendet werden.

   d. `c`: fahre mit der Ausführung des Programm fort

   e. `n`: Ausführung einer Programmzeile; besteht dieser aus einem Funktionsaufruf *bleibt* der Debugger *in der ersten Zeile* der Funktion *stehen*.

   f. `s`: Ausführung einer Programmzeile; besteht dieser aus einem Funktionsaufruf so *überspringt* der Debugger die Funktion.

   g. `Ctrl c`: Stoppt die Ausführung des aktuellen Programmes.

   h. `backtrace`: Zeigt den Callstack der letzten Aufrufe an. Dieser Befehl funktioniert auch nach einem Programmabsturz und kein sehr hilfreich

[56] Befehle, welche zum Start des Debuggers ausgeführt werden sollen (z.B. „remote target ...“) können im File `.gdbinit` (den *Punkt* beachten) abgelegt werden (wichtig: ein `Carriage Return` am Ende des Files scheint Probleme zu machen). Das File sollte im Home Verzeichnis (`/home/oscar`) abgelegt werden.

sein, die Absturzursache zu identifizieren.

6. Beenden des Debuggings:
   Um den Debugger zu beenden genügt es in der Regel auf der Kommandozeile den Befehl

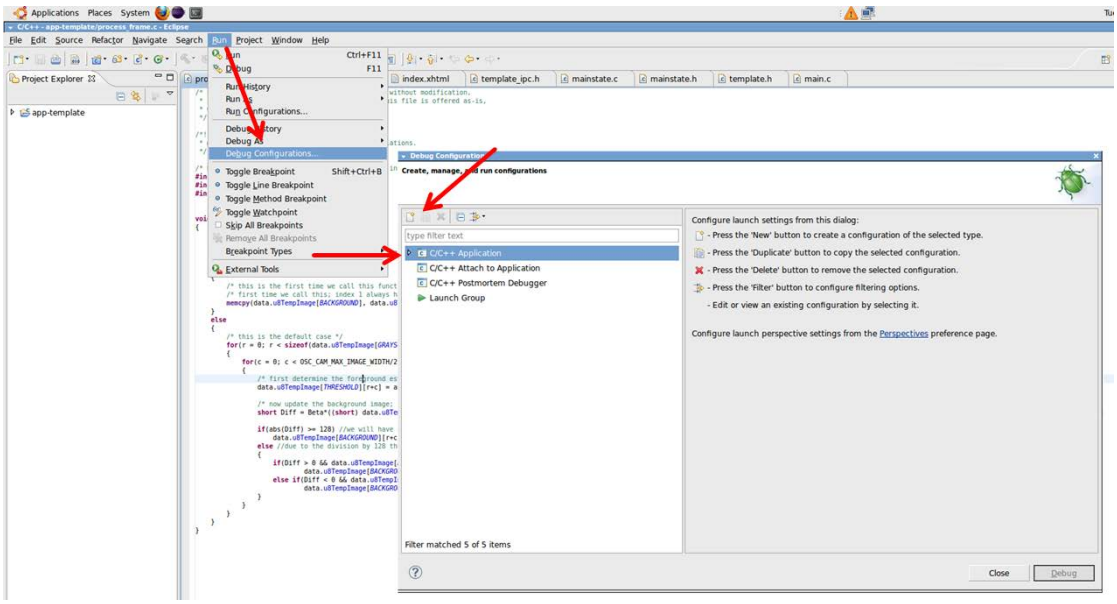
   `quit`

   einzugeben. Damit wird der Debugger geschlossen und in der Regel stopp auch der `gdbserver` seine Arbeit. Falls es bei einem erneuten Versuch, eine Debugging Session zu starten, Probleme gibt, so kann dies daran liegen, dass der `gdbserver` auf dem leanXcam Target noch am Laufen ist. Ggf. kann dies mit `top` verifiziert werden und der Prozess mit `kill` beendet werden.

8.3.1. Debugging aus dem Eclipse IDE

Ebenso wie für die Entwicklung eines C/C++ Programmes unter Linux bietet das Eclipse IDE eine GUI Alternative zum Kommandozeilen-basierten Debugging mittels gdb. Hierzu ist das Eclipse IDE geeignet zu konfigurieren:

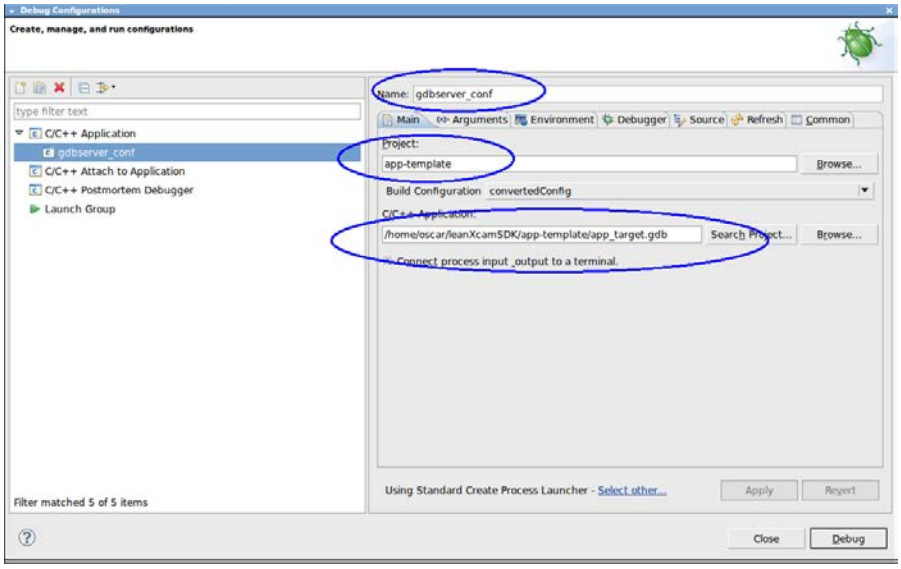
1. Im Eclipse `Run -> Debug Configurations -> C/C++ Application` auswählen und New launch cofiguration wählen.



2. In dem Dialogfenster sind – von oben nach unten – die folgenden Einstellungen vorzunehmen:
   a. Ein Name für die neue Debug Konfiguration

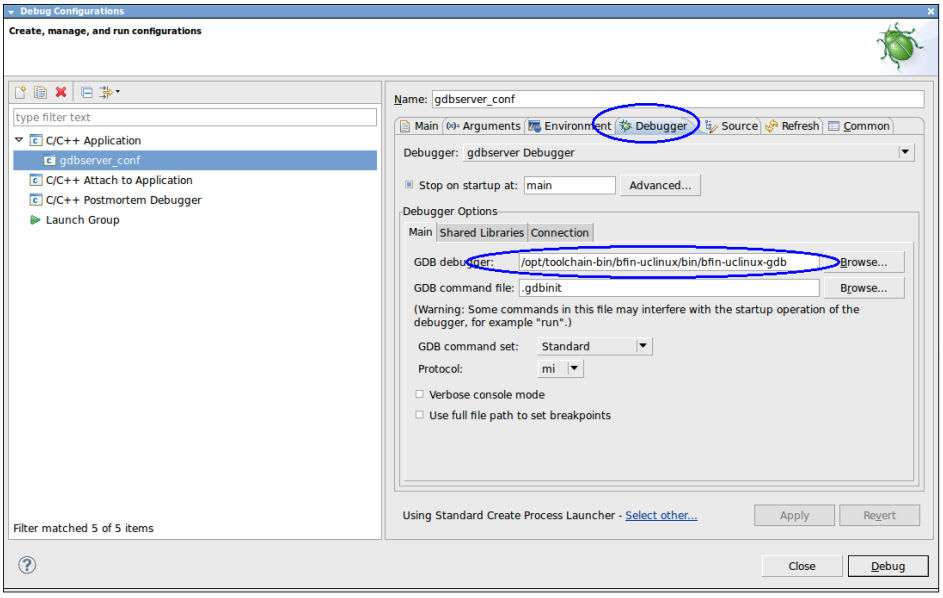
   b. Das betreffende Projekt

   c. Die Applikation, welche dem Debugger beim Start übergeben werden muss: dies ist das Binärfile `app_target.gdb` für das Blackfin Target, welches auch beim Start des `gdb`-Debuggers als Argument über geben wird (s. Punkt 3 oben)



3. Als nächstes muss im Tab `Debugger` des Dialoges der Pfad zum Blackfin Debugger angegeben werden. Dieser liegt in der Standard leanXcam VM unter:

   `/opt/toolchain-bin/bfin-uclinux/bin/bfin-uclinux-gdb`



   Unter dem entsprechenden Eingabefeld für den Debugger ist noch das File für etwaige `gdb` Kommandos angegeben[57]:

   `.gdbinit`

   Erstellen Sie dieses File im Homeverzeichnis (`/home/oscar`) und fügen Sie an erster Stelle den folgenden Befehl ein[58]:

   `target remote 192.168.1.10:7777`

[57] Den Punkt an erster Stelle des Filenamens beachten.
[58] Dies ist die “normale” Syntax für den `gdb` Debugger. Wahlweise können auch noch andere Befehle wie häufig verwendete Breakpunkte etc. eingegeben werden.

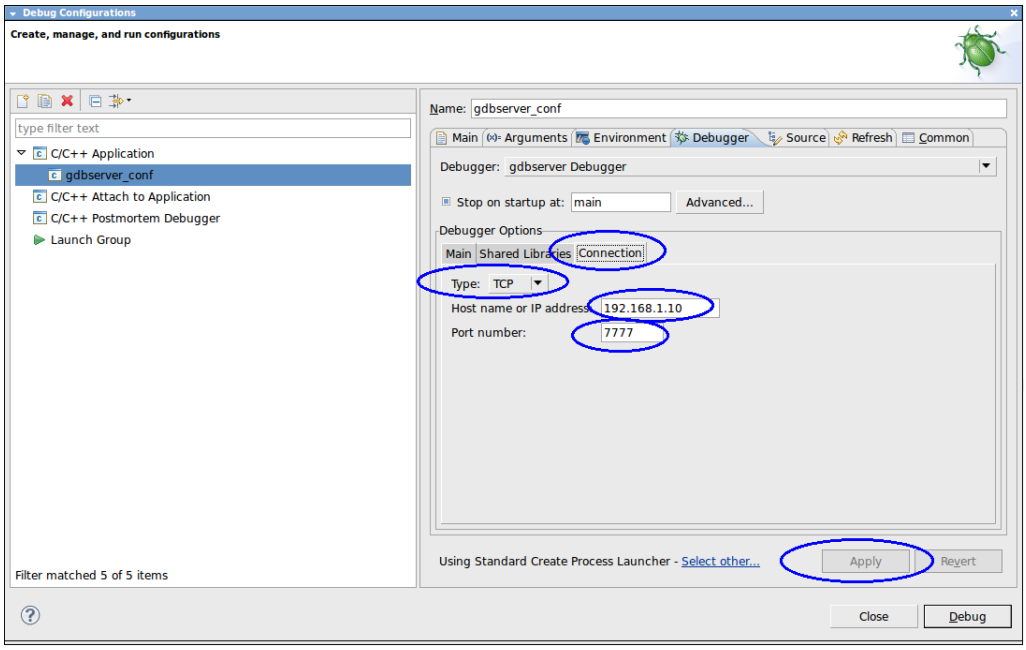
Damit wird der Debugger beim Start direkt mit dem gdb-Server verbunden.

4. Dann im Dialog unter dem Tab `Connection` – wieder von oben nach unten – die folgenden Settings vornehmen:

   a. Verbindungstyp: TCP

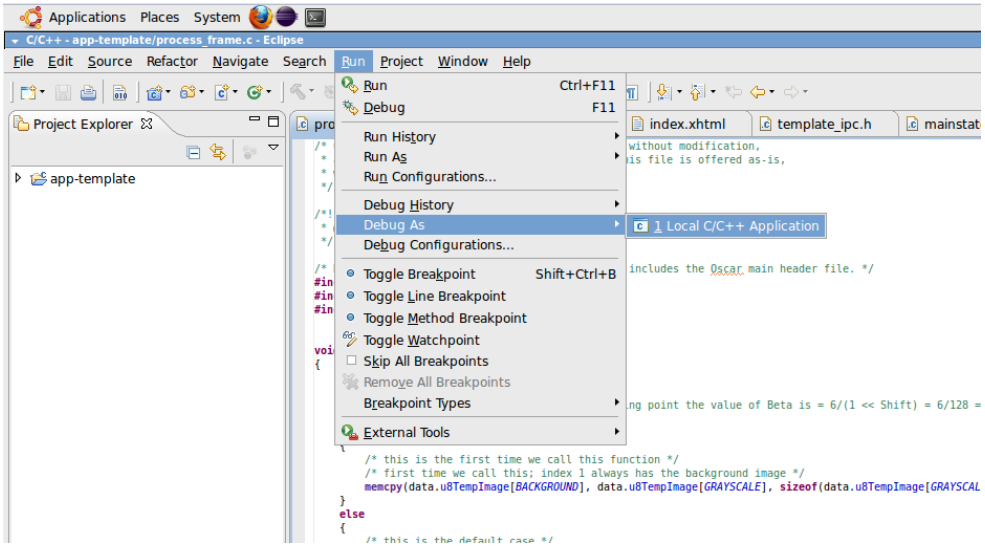
   b. IP Adresse: 192.168.1.10

   c. Port Nummer: 7777



5. Nach dem Drücken des `Apply` Buttons ist ein Debugging der Anwendung möglich mittels (vorher den `gdbserver` auf dem Target starten, s. Punkt 2):

   `Run -> Debug As -> Local C/C++ Application`



9. Annex

9.1. References

[1] http://www.scs-vision.ch/de/leanxcam/index.html
[2] https://github.com/scs/leanXcam/wiki
[3] https://github.com/scs?tab=repositories
[4] http://de.wikipedia.org/wiki/Classless_Inter-Domain_Routing
[5] http://www.boa.org
[6] https://www.virtualbox.org/wiki/Downloads
[7] http://www.scs-vision.ch/de/leanxcam/hardundsoftware.html
[8] http://git-scm.com/doc
[9] http://www.stack.nl/~dimitri/doxygen
[10] https://github.com/scs/leanXcam/wiki/Downloads
[11] http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Standards.html
[12] http://www.ijon.de/comp/tutorials/makefile.html
[13] TA.BA_EBV.F1301_Skript_V-1-0.pdf
[14] http://www.dedoimedo.com/computers/virtualbox-network-sharing.html
[15] https://github.com/scs/leanXcam/wiki/Users-Guide-Board-connectivity
[16] http://de.wikipedia.org/wiki/Public-Key-Authentifizierung

9.2. Abkürzungen

Lise 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

9.3. Direkte Installation auf einem Linux Betriebssystem

Falls der Host Computer schon ein Linux Betriebssystem beinhaltet und die Installation der VirtualBox SW inklusive der leanXcam VM vermieden werden soll, so kann versucht werden, das SDK direkt auf dem Host zu installieren[59].

Für die Installation kann das gesamte Framework – inklusive Toolchain für die Erstellung des Codes auf der leanXcam – von einem git-Repository heruntergeladen werden (s. auch Abschnitt 4.2.1). Um dies zu vereinfachen, findet sich im SW Verzeichnis auf ILIAS (Abbildung 3) das Skript `get_git-repositories.sh` , welches alle notwendigen Dateien in das lokale Verzeichnis herablädt.

[59] Hierzu sollten Sie über einige Grundkenntnisse des Linux Betriebssystems verfügen.

Danach müssen Sie nur das OSCar Framework gemäss Abschnitt 4.2.1 kompilieren und den Pfad zur Toolchain zu Ihrer Pfadvariablen hinzufügen. Am besten machen Sie dies im File

`.bashrc`

damit die Einstellungen bei jedem Login automatisch vorhanden sind:

```
export PATH=$PATH:~/leanXcam/leanXcamSDK/toolchain-bin/bfin-elf/bin:~/leanXcam/leanXcamSDK/toolchain-bin/bfin-linux-uclibc/bin:~/leanXcam/leanXcamSDK/toolchain-bin/bfin-uclinux/bin
```

Die Anweisung darf natürlich keine Zeilenumbrüche im File enthalten.
Jetzt können Sie die Installation testen, indem Sie eines der Beispiele erstellen. Dabei sollte Sie auch immer zuerst den `./configure` Befehl ausführen, um den Pfad zum OSCar Framework anzupassen.

9.4. git in a Nutshell

Für eine sehr gute Einführung in `git` sehen Sie unter [8] 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[60].

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'`.

[60] Falls die Meldung “warning: remote HEAD referes to nonexistent ref, unable to checkout.“ erfolgt, so genügt es, einen Checkout auf eine neuen – selbst definierten – Branch durchzuführen.

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`[61].

11. `git branch:`
    Zeige aktuelle Braches 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`[62] 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`[62] 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:`
    Der Branch `master` wird vom angegebenen (`github`[62] Remote) Repository (oder vom „origin“, d.h. vom Repository, welches mit „clone“ kopiert wurde[63]) 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`[62] 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 angegebene Hash `sha1` wird in den aktuellen Branch eingefügt (merge). Allerdings führt dies zu einem „detached HEAD[64]“.

17. `git reset –-hard rev:`
    Setze den Zustand des Repositories zurück auf `rev` und lösche alle Änderungen, welche seither durchführt wurden.

18. `git revlog`:
    (und Verwalte) die Historie der Änderungen am Repository.

[61] Wir nehmen an, dass wir aktuell auf dem Branch `release` arbeiten.
[62] Die Syntax der URL variiert je nach Server.
[63] Die Definition von „origin“ findet sich in `.git/config`.
[64] 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.

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 angegeben URL. D.h. alle
    `git push/pull origin`
    Befehle erfolgen dann bez. dieses Remote Repositories. Der aktuell wird wir so angezeigt:
    `git remote -v`

9.4.1. 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`[65] 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[66]:
   Wechseln Sie in das Verzeichnis ~/.ssh und geben Sie auf der Kommandozeile folgenden Befehl ein:
   `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)“.

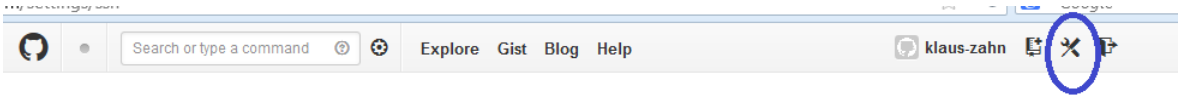
   ```
   Enter passphrase (empty for no passphrase):
   Enter same passphrase again:
   Your identification has been saved in /home/oscar/.ssh/id_rsa.
   Your public key has been saved in /home/oscar/.ssh/id_rsa.pub.
   The key fingerprint is:
   dc:4c:b3:d0:7f:9b:60:f8:87:cc:ce:dc:fc:82:be:72 email
 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

   Danach erscheint die Frage nach einem Password, 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.

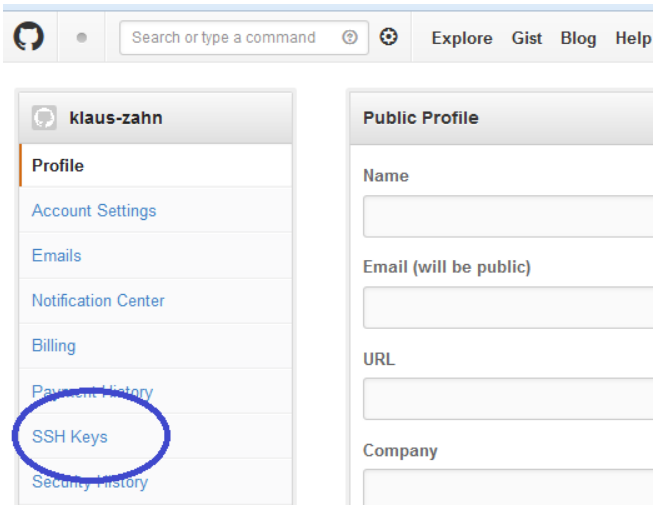
2. Registrieren des private Key in `github`:
   Für eine authentifizierte Verbindung mit `github` (Voraussetzung für einen Schreibzugriff) muss der private Key auf `github` hinterlegt werden. Hierzu in

[65] https://github.com/
[66] Falls ein Key Paar schon existiert, kann dieser Schritt übersprungen und direkt zu Punkt 2. übergegangen werden.

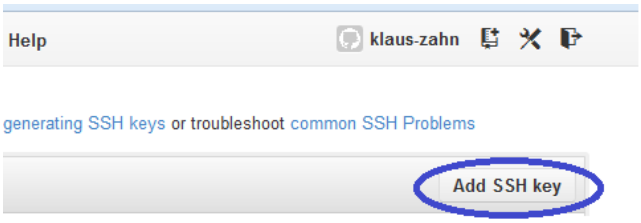
`github` unter `Account settings` (blaue Markierung)



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 9.4 unter den Punkten 12 und 13 angegebenen Befehlen die Inhalte des lokalen Repositories in das Remote Repository übertragen werden.

9.5. Trouble Shooting

9.5.1. 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 ↑↓ navigieren, sowie mittels Eingabe von Slash

```
/
```

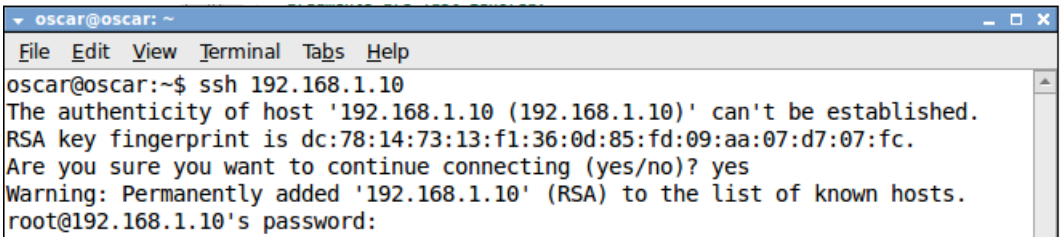
nach einem Pattern suchen (mit jedem Drücken der n-Taste springen Sie zum nächsten Match).
Die Manual Page verlassen Sie mit der q-Taste.

9.5.2. Issues bei der SSH-Verbindung mit der leanXcam

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 [16], 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[67]. 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 the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
dc:78:14:73:13:f1:36:0d:85:fd:09:aa:07:d7:07:fc.
Please contact your system administrator.
Add correct host key in /home/oscar/.ssh/known_hosts to get rid of this
message.
Offending key in /home/oscar/.ssh/known_hosts:1
RSA host key for 192.168.1.10 has changed and you have requested strict
checking.
Host key verification failed.
```

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 leanXcam HW erstellt wurde. Um das Problem zu lösen, einfach das File `known_hosts` löschen:

[67] Damit wird die Authentizität des Ziel Computers manuell bestätigt.

```
rm ~/.ssh/known_hosts
```

9.5.3. Gemeinsamer Ordner nicht verfügbar

Zuerst einen Reboot der leanXcam-SDK 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 12 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)
lrm 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 12 nochmals durchführen. Ggf. muss ein neues Verzeichnis auf dem Host gewählt werden.

9.5.4. 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
```

9.5.5. 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

9.5.6. Probleme mit Netzwerkverbindungen der VM nach Reboot

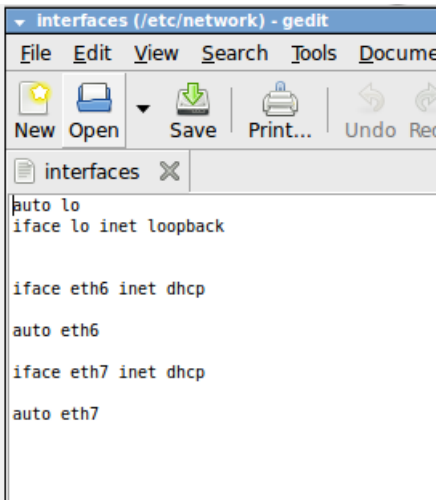
Verifizieren Sie die korrekte Konfiguration der Netzwerk Interfaces im File

`/etc/network/interfaces`

Editieren Sie das File (mit `sudo`) gemäss:

```
sudo gedit /etc/network/interfaces
```

und verifizieren Sie, dass der Inhalt wie folgt aussieht:



Im Folgenden als Text für Copy/Paste:

```
auto lo
iface lo inet loopback


iface eth6 inet dhcp

auto eth6

iface eth7 inet dhcp

auto eth7
```

Speichern Sie das File und starten Sie die Netzwerk Interfaces neu:

```
sudo /etc/init.d/networking restart
```

9.5.7. 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 [6] heruntergeladen werden. Die Installation erfolgt durch eine Doppelklick auf das File.
Nach der Installation des Extension Pack die leanXcam VM herunterfahren und gemäss Abbildung 24 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 25 gezeigt verbunden werden. Ein Memory Stick erscheint dann als „Media“ auf dem Desktop und sollte unter `/media` gemounted sein.

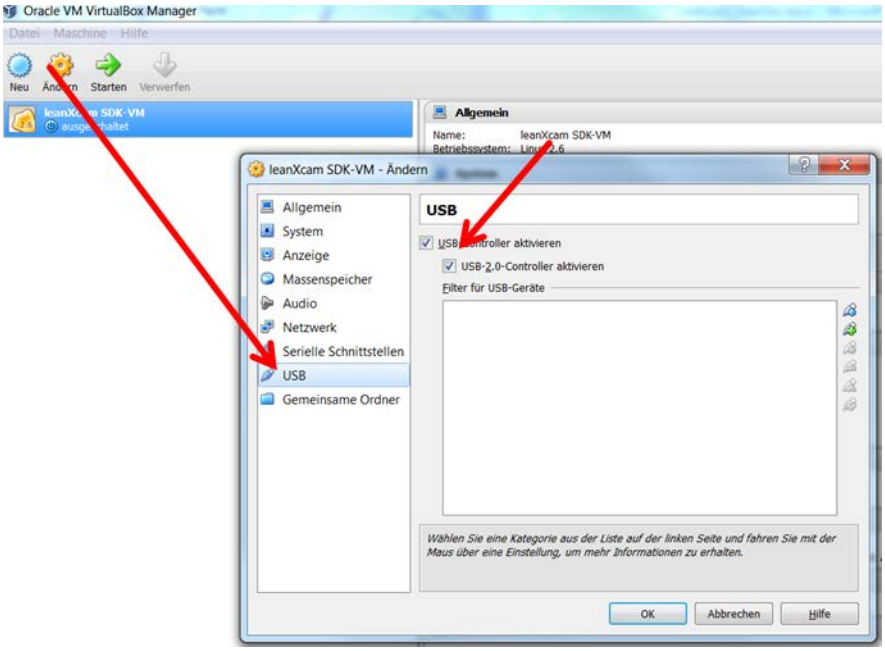


Abbildung 24: *Setzen der USB Einstellungen der VM.*

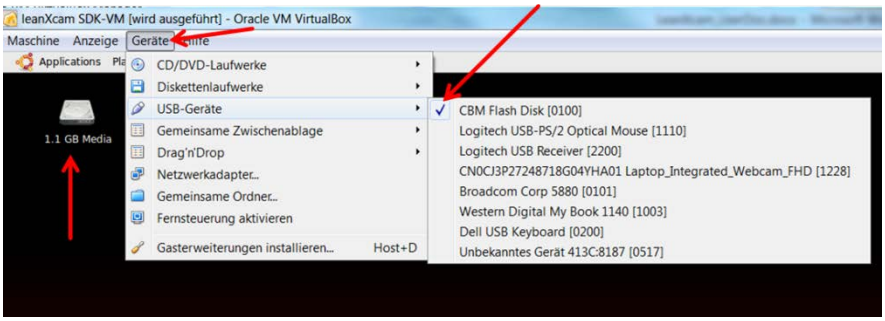


Abbildung 25: *Verbinden eines UBS Devices (hier ein Memory Stick).*

9.5.8. 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 26, Bemerkung unter der Abbildung beachten), die leanXcam-SDK VM löschen (d.h. gemäss Abbildung 26 vorgehen, aber `Alle Dateien entfernen` auswählen) und dann die leanXcam-SDK VM erneut gemäss Kapitel 3 importieren.

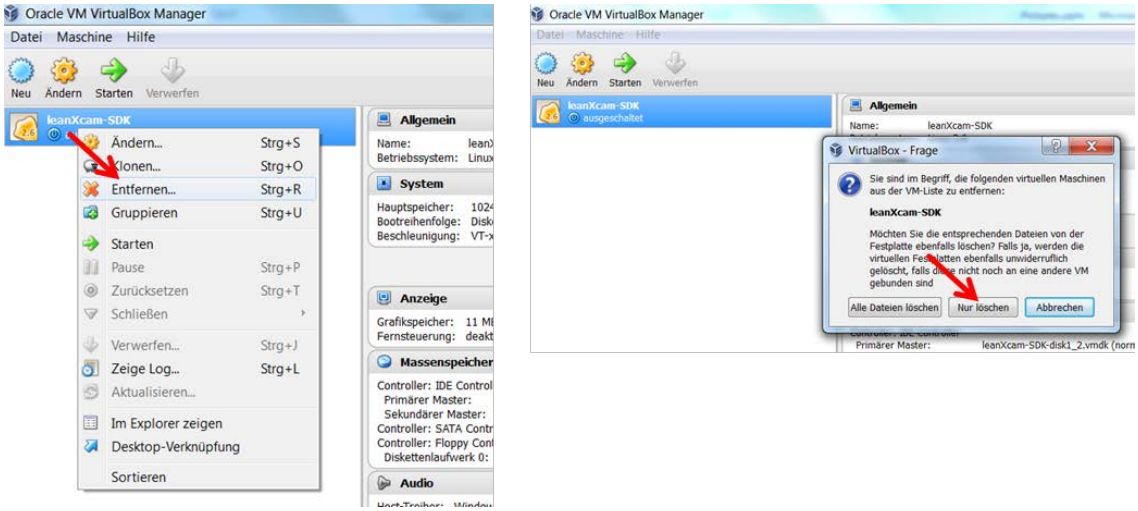


Abbildung 26: *Entfernen einer VM.* ***Vorsicht:*** *`Alle Dateien entfernen` löscht auch die Files auf der Festplatte.*

9.5.9. 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: