Hochschule Esslingen
University of Applied Sciences

Team Project

Quadrocopter

in the degree course ASM-SB
of the Faculty Graduate School
ASM2

Oliver Breuning
Martin Brodbeck
Jürgen Schmidt
Phillip Woditsch

Periode: Summersemester 2015
Professor: Prof. Dr. Jörg Friedrich

Contents

List of Figures

1

1 Basics

This chapter deals with the basics which are needed for the quadrocopter project. Beginning with the Virtual machine and the running Operating system Ubuntu in the Virtual machine. After that a description of the Toolchain follows. Finally a description of Eclipse including the settings of the remote debugging closes this documentation.

1.1 Development environment

1.1.1 VM

This project uses the VMWare player of the company VMWare [1]. This gives the ability that no dependency of a special operating system exists. It can run on any system like mac, windows or linux.

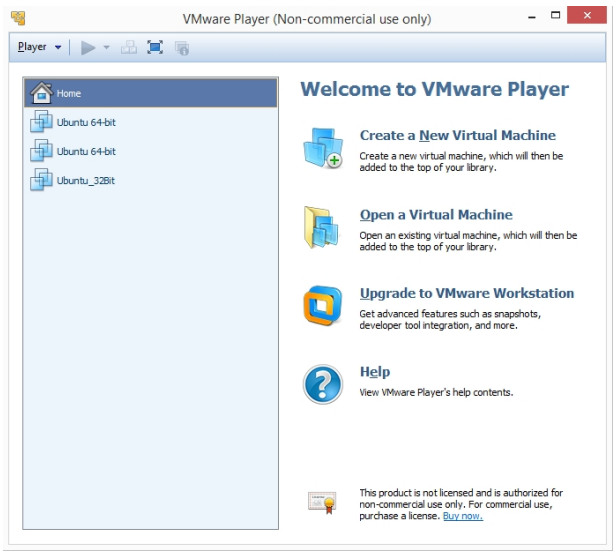

Figure 1.1: VMWare Player

1 https://my.vmware.com/de/web/vmware/free#desktop_end_user_computing/vmware_player/7_0

1.1.2 Ubuntu

Here the actual Ubuntu 14.10 which is installed with the help of the VMWare player is used. For programming the software eclipse is taken. First an actual image of Ubuntu needs to be downloaded:

```
1 http://www.ubuntu.com/download
```

Figure 1.2 shows the the window of the VMWare player directly after starting. To add a new virtual machine a click on 'Create a New Virtual Machine' has to be made.

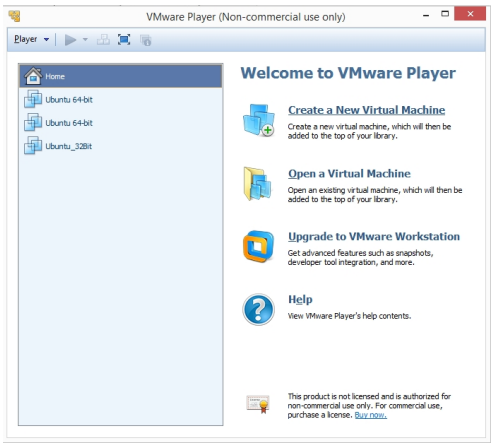

Figure 1.2: Step 1 - Installing Ubuntu

Then under the 'Installer disc image file (iso)', the path to the before downloaded ISO-file needs to be set like in the figure 1.3.

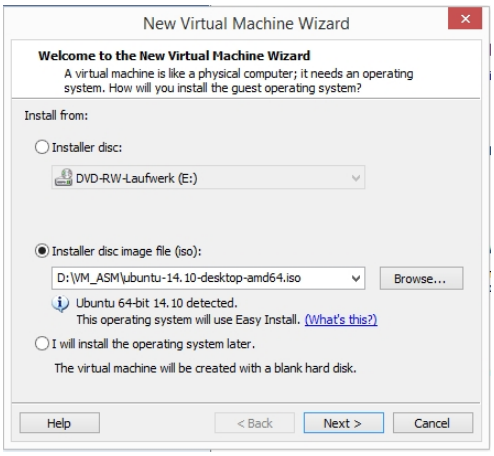

Figure 1.3: Step 2 - Installing Ubuntu

Figure 1.4 shows the next step. A full name, a user and a password is set. In this project 'user' with password 'user' is used.

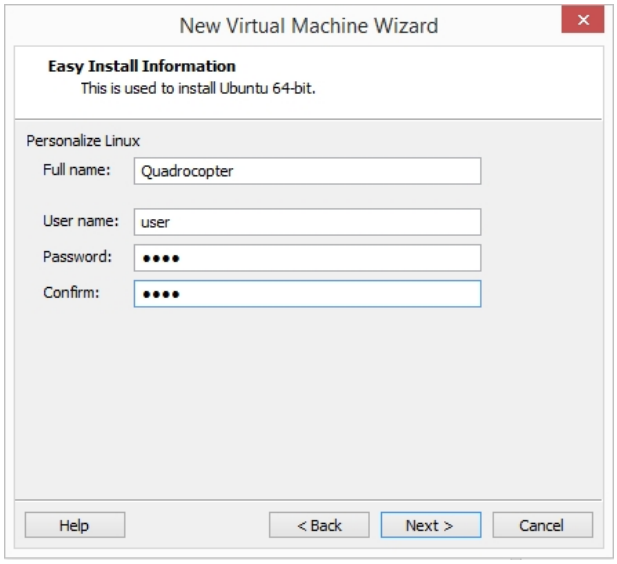

Figure 1.4: Step 3 - Installing Ubuntu

Then a name for the virtual machine and the location, where the virtual machine is stored is chosen. See figure 1.5.

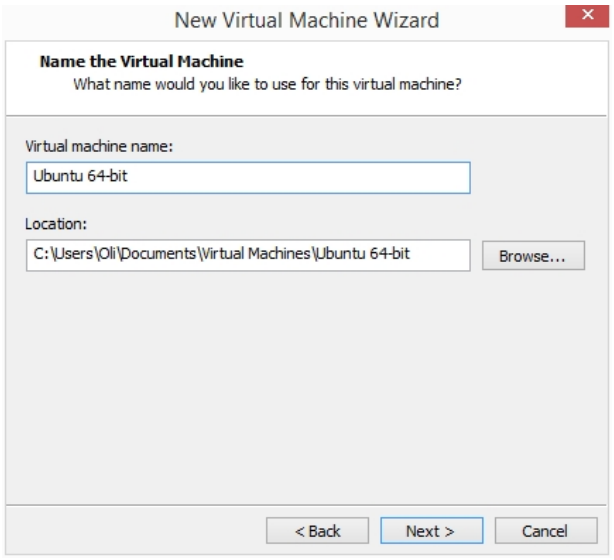

Figure 1.5: Step 4 - Installing Ubuntu

The following figures 1.6 and 1.7 show the option for the size and the splitting and finally a summary.

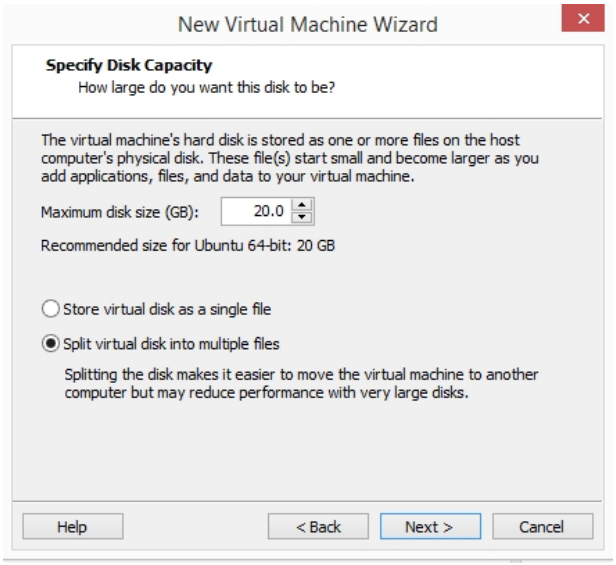

Figure 1.6: Step 5 - Installing Ubuntu

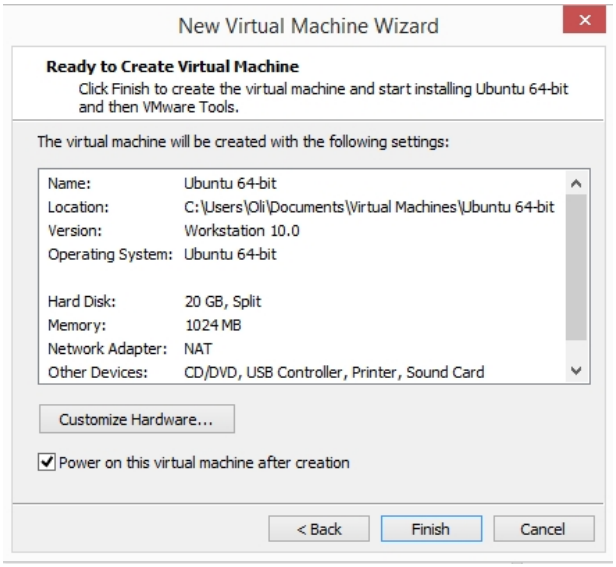

Figure 1.7: Step 6 - Installing Ubuntu

Then VMWare tools wants to be installed. This tool is needed for example to enable dynamic rescaling of the Desktop of Ubuntu within the VMWare player, so it should be installed. See figure 1.8.

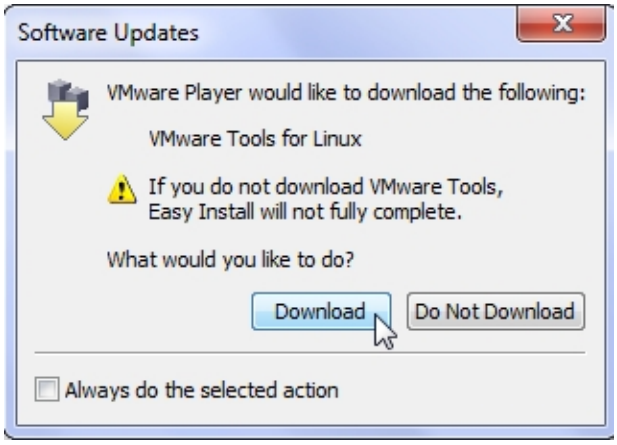

Figure 1.8: Step 7 - Installing Ubuntu

Finally the login window of Ubuntu (figure 1.9).

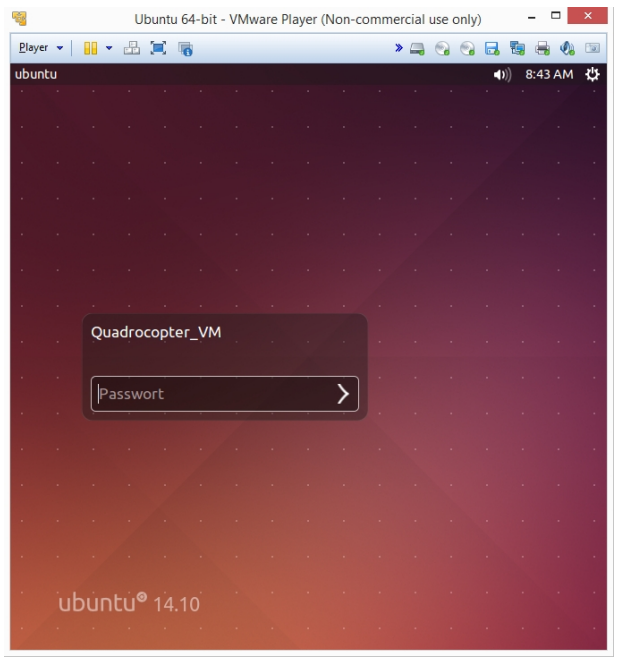

Figure 1.9: Step 8 - Installing Ubuntu

Remark:
If the rescaling later on will not work, then after the start and login into Ubuntu, the following command needs to run in the terminal window:

```
1 sudo apt-get install open-vm-tools-desktop
```

1.1.3 Toolchain

A Toolchain enables compilation on the fast development computer instead on the Raspberry Pi. This is called cross compiling. To make it possible, following commands have to be called in the Ubuntu operating system which is running in the virtual machine. This description is based on the blog entry from Halherta [1].
First the cross compiling toolchain will be downloaded and set up. The following commands will be called in a terminal window.
The following command downloads the native build tools and the GIT tool which will be used to download the cross compiling toolchain.

```
sudo apt-get install build-essential git
```

Next a directory is created and then the directory is opened.

```
mkdir rpi
cd rpi
git clone git://github.com/raspberrypi/tools.git
```

The last command downloads the official cross compiling Toolchain. In the folder /home/user/rpi/tools/arm-bcm2708 are now four different folders stored which contain different toolchains.

```
 arm-bcm2708-linux-gnueabi
 arm-bcm2708hardfp-linux-gnueabi
 gcc-linaro-arm-linux-gnueabihf-raspbian
 gcc-linaro-arm-linux-gnueabihf-raspbian-x64
```

Due to the reason that the used Ubuntu is a 64 bit OS the fourth toolchain has to be used. To enable compilation also directly from the command line, the path to the Toolchain has to be added in the PATH variable of the Ubuntu system. To do so, the following actions need to be done.

```
cd ~/
nano .bashrc
export PATH=$PATH:$HOME/rpi/tools/arm-bcm2708/
   gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin
source .bashrc
```

The last command updats the PATH variable.
If all actions are successfully done, it can be proved by:

```
arm-linux-gnueabihf-gcc -v
```

1 http://hertaville.com/2012/09/28/development-environment-raspberry-pi-cross-compiler/

If all works fine the terminal window should show following output.

Figure 1.10: Prove of successfull working Toolchain

1.1.4 Eclipse

Then the Ubuntu system is started. Next the actual 64 Bit Eclipse IDE for C/C++ Developers, is downloaded from the webpage of eclipse within the Ubuntu system:

```
1 https://www.eclipse.org/downloads/?osType=linux
```

The next steps are made in the terminal of the first set up Ubuntu system. First the current Java runtime environment is installed.

```
1 sudo apt-get install openjdk-7-jre
```

Then the before downloaded eclipse is extracted and stored to /opt

```
1 sudo tar -xvzf eclipse-cpp-luna-SR2-linux-gtk-x86_64 -C /opt/
```

In order to access the most recent eclipse, a custom desktop link has to be created and edited.

```
1 sudo touch /usr/share/applications/eclipse-recent.desktop;
2 sudo gedit /usr/share/applications/eclipse-recent.desktop &
```

An editor will open after executing the above shown lines. Paste the following lines in order to create a valid Ubuntu desktop launcher:

```
1 [Desktop Entry]
2 Type=Application
3 Name=Eclipse Luna
4 Comment=Eclipse Integrated Development Environment
5 Icon=/opt/eclipse/icon.xpm
6 Exec=/opt/eclipse/eclipse
7 Terminal=false
8 Categories=Development;IDE;Java;
```

Optionally, a starter icon can be put to the desktop of Ubuntu by copying the above created Ubuntu desktop launcher:

```
1 sudo cp /usr/share/applications/eclipse-recent.desktop ~/Desktop/
```

1.1.4.1 Remote Debugging

Enabling remote debugging is really helpfull when a written program did not do what is expected. To make it possible, some configurations in the Debug Configurations need to be done. This window can be found under 'Run/Debug Configurations...' In the starting window a click with the left mousebutton on 'C/C++ Remote Application' on the left part of the window needs to be done. Then click on 'New'. See figure 1.11.

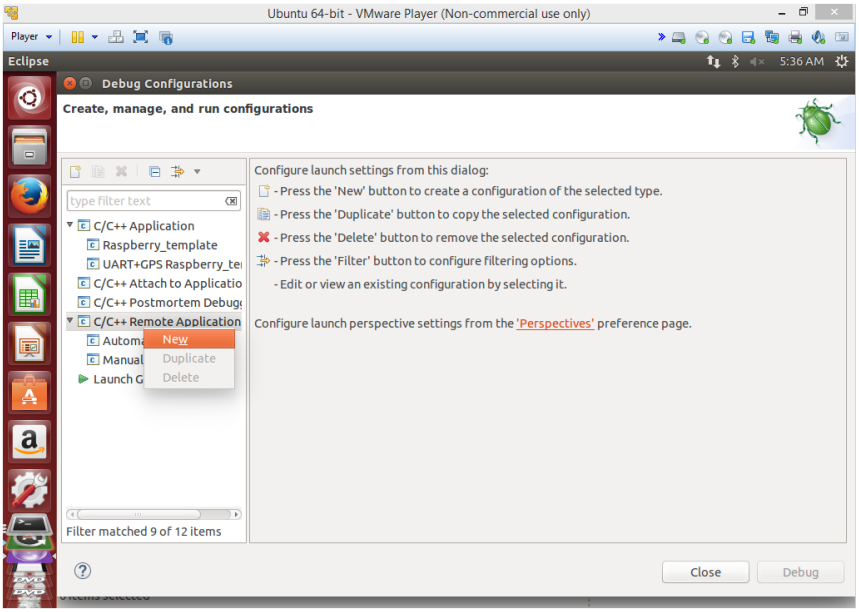

Figure 1.11: Making new Debug Configuration

In the appearing window write down first the Name 'Automatic_Debug' and then in the tab 'Main' (see figure 1.12) write down the name of the project in the textfield 'Project:' and in 'C/C++ Application:' the executable, the binary file, which is in the Debug folder of the project. Under 'Build configuration' 'Use Active' and the radio button 'Enable auto build' has to be activated. When clicking under 'Connection:' on 'New...' a new window, like in figure 1.13 appears. Here 'SSH only' and 'Next>' needs to be clicked. After that, in the next automatically popped up window all what needs to be written down can be seen in figure 1.14, and then it has to be confirmed by clicking on 'OK'. Last but not least there needs to be set a Folder which can be a existing or a new one in 'Remote Absolut File Path for C/C++ Application:' and under 'Commands to execute before application' the commands

```
sudo -i
chmod +x Raspberry_Template
```

have to be included. Last make sure that in the lower area of the window 'Using GDB (DSF) Automatic Remote Debugging Launcher' is active.

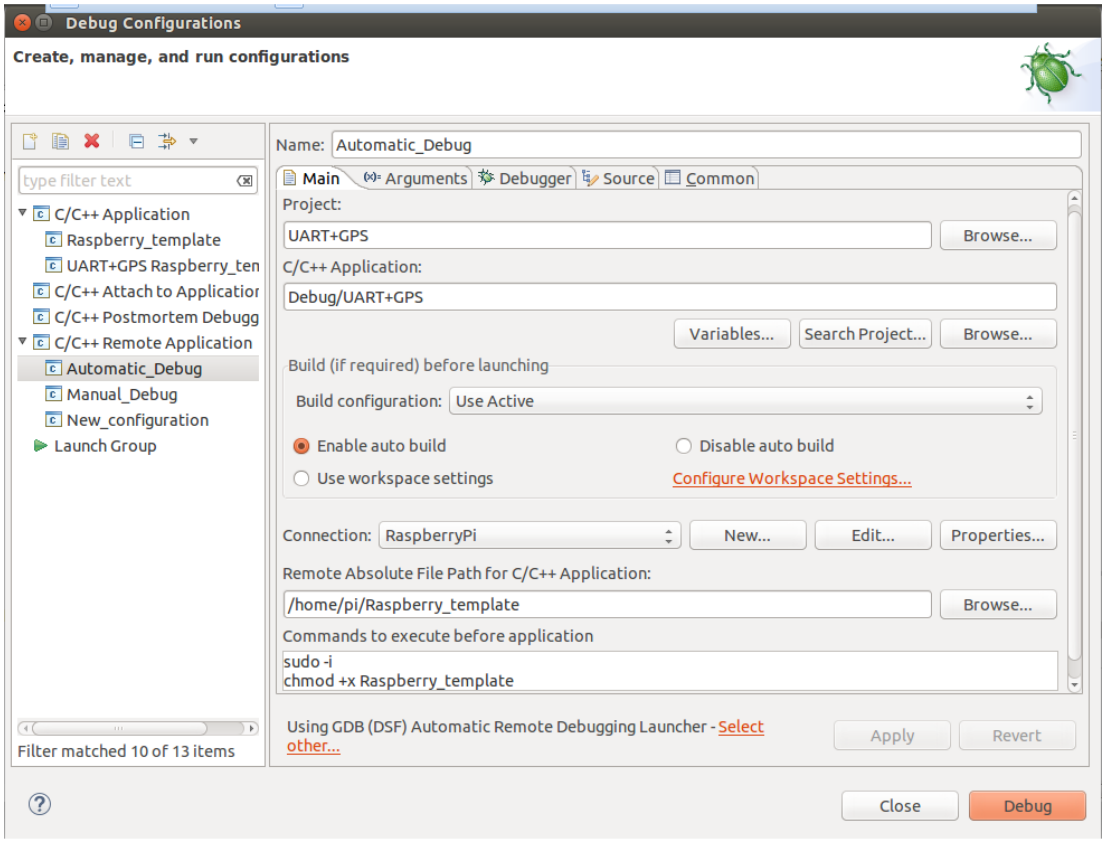

Figure 1.12: Debug Configuration Window - Main tab

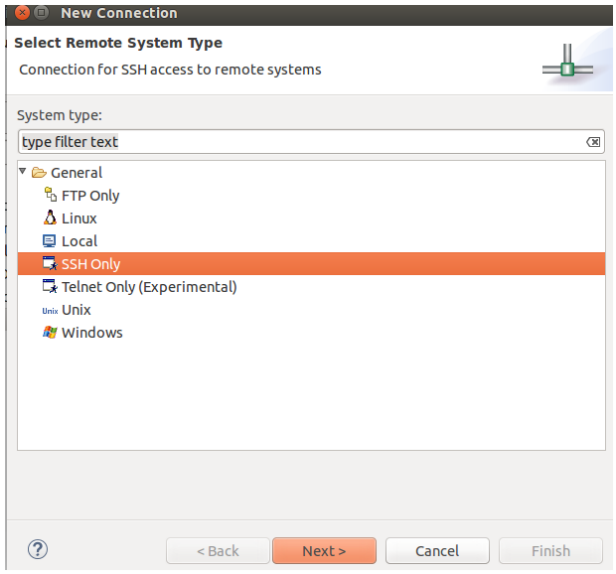

Figure 1.13: Creating new connection 1

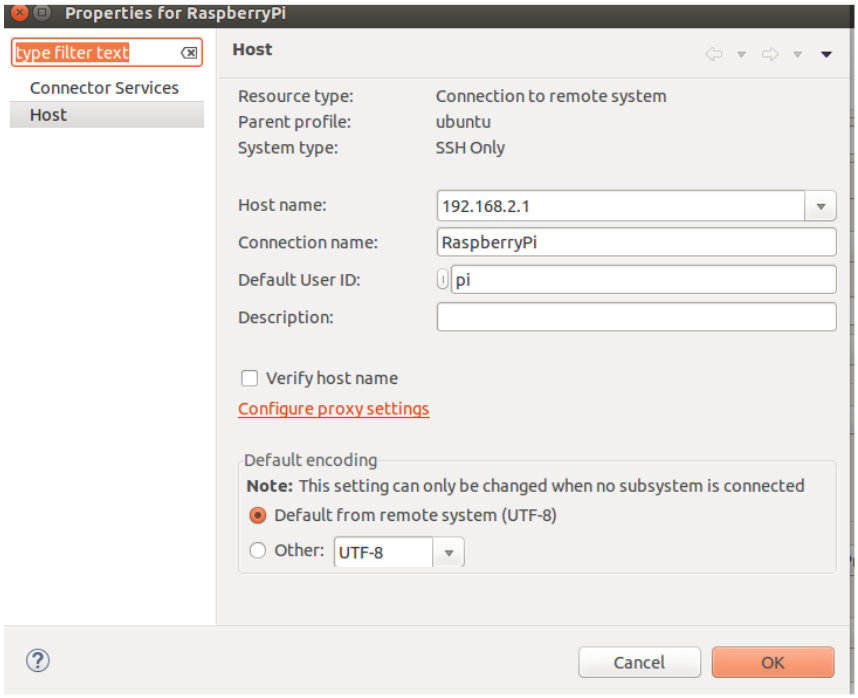

Figure 1.14: Creating new connection 2

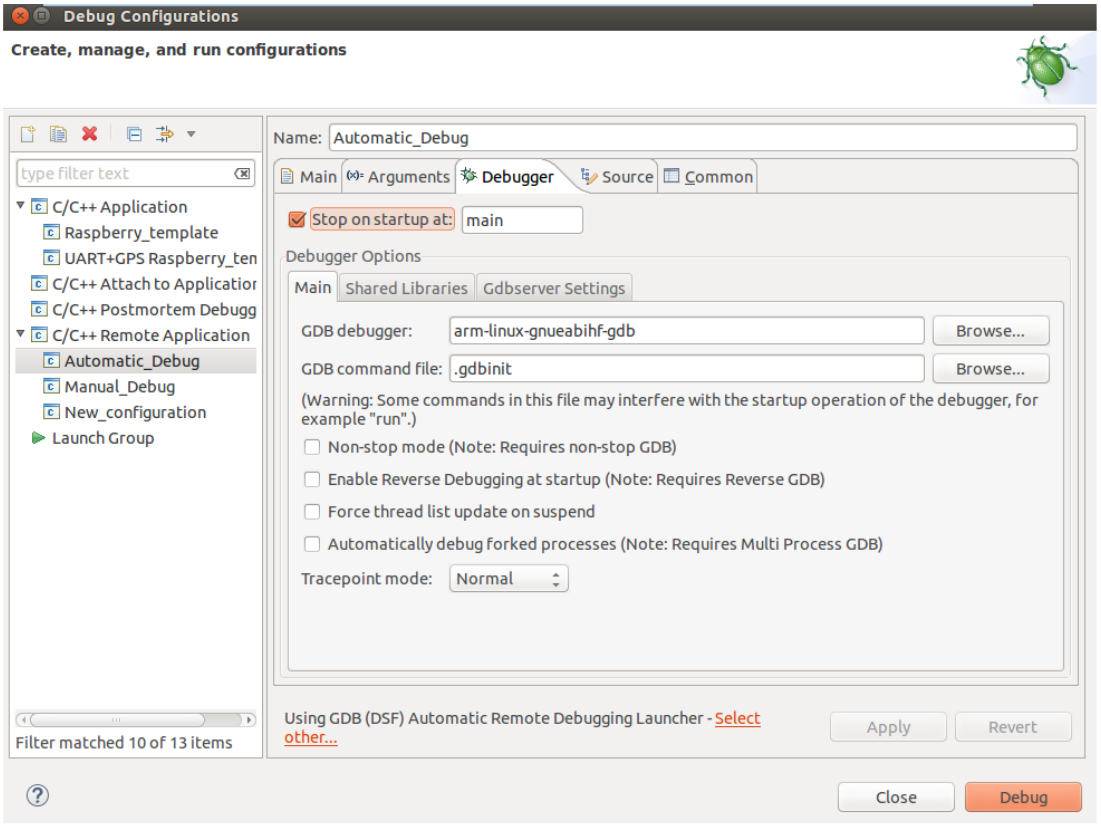

Figure 1.15: Debug Configuration Window - Debugger tab

After this is finished, in the Debugger tab, there needs in the 'GDB debugger' textfield 'arm-linux-gnueabihf-gdb' written down. This ensures that the ARM toolchain connects to the GDB server on the Raspberry (See figure 1.15).

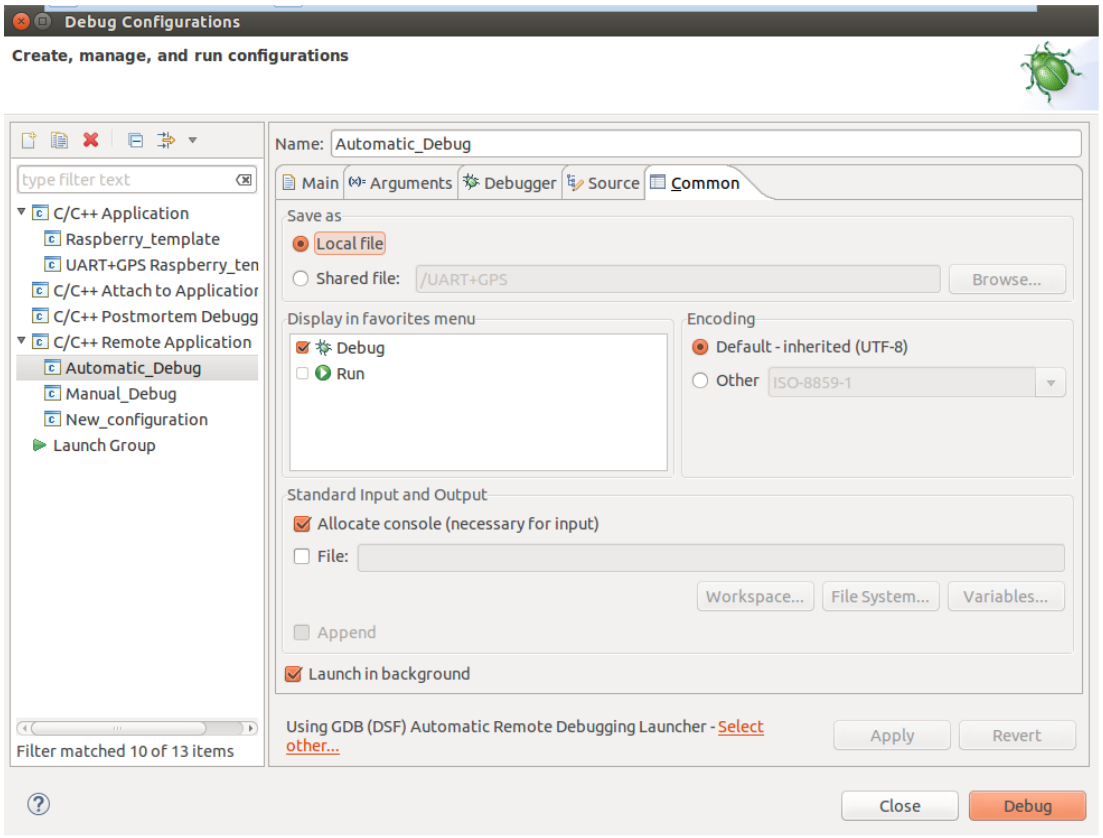

Figure 1.16: Debug Configuration Window - Common tab

Then in the 'Common' tab in 'Display in favorites menu' the 'Debug' checkbox needs to be activated. After that the window can be closed. Now all necessary actions are done and an existing project can be by debugged by clicking on the small arrow beneath the the small bug in the menu bar. Here the before saved Configuration (Automatic_Debug) has to be activated what the debugging directly starts.

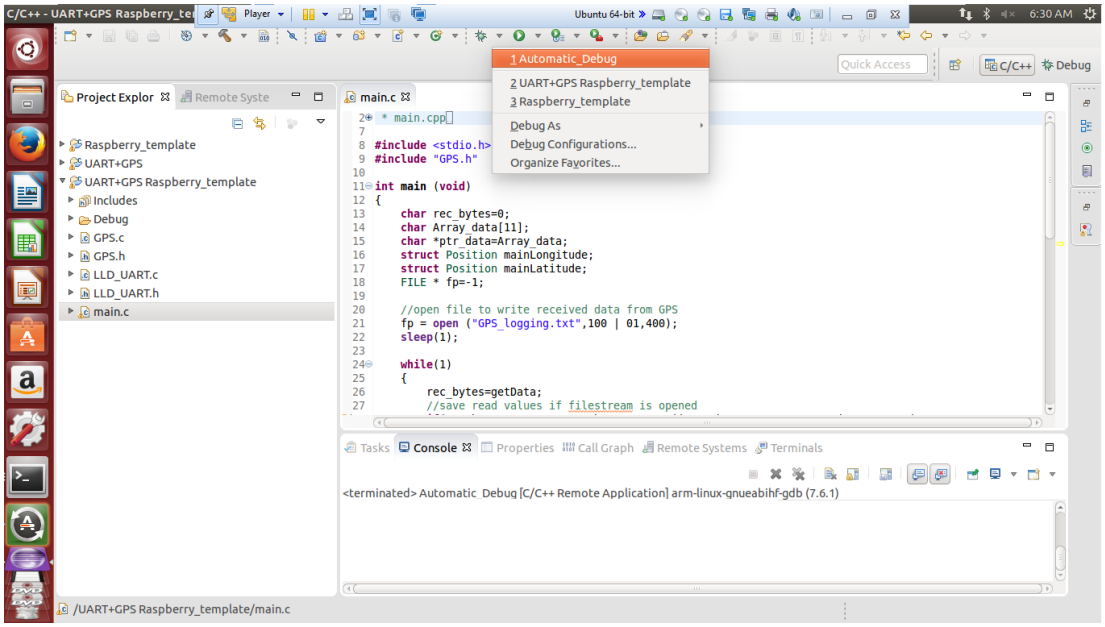

Figure 1.17: Start Debugging