

Firmware Development

Last edited by **AndreR** 6 days ago

How to Use the TIGERs Firmware

This How-To will show you the required steps to setup your IDE, compile the TIGERs source code for the robot and our base station, and to debug it.

System Requirements

Any windows or linux machine can be used. We tested Windows 7 x64, Windows 8.1 x64, Windows 10 x64, and Arch Linux.

IDE Setup

We use the Eclipse CDT environment for development. The instructions have been tested with Eclipse 2019-06.

1. Download Eclipse CDT for your platform: [2019-06 CDT](#)
2. Unpack the CDT and start Eclipse
 1. Select a workspace. It should not contain any whitespaces!
3. Install the [GNU MCU Eclipse](#) plug-in
 1. In Eclipse select: Help => Install New Software...
 2. In the "Work with" field enter: <http://gnu-mcu-eclipse.netlify.com/v4-neon-updates> and hit enter
 3. Select at least the packages labeled "GNU MCU ARM C/C++ Cross Compiler" and "GNU MCU C/C++ OpenOCD Debugging"
 4. Click next, accept the licence, click finish. During the installation confirm the installation of unsigned content
 5. Alternatively, you may also install the plugin over the Eclipse Marketplace, search for "GNU MCU"
 6. Restart Eclipse when prompted

ARM Toolchain

The ARM Toolchain contains the compiler, linker and debugger for the embedded microcontrollers. The recommended and tested version is GCC 6.0.

There are known issues with GCC 7 and newer.

1. Download the appropriate toolchain for your platform (I recommend the zip or tar.bz variant) from the [GNU ARM Embedded](#) project.
2. Unpack the toolchain to a location of your choice. Again, avoid whitespaces.

Configure Toolchain

1. In eclipse open: Window => Preferences
2. Select: MCU => Global ARM Toolchains Paths
3. Next to the toolchain folder, click on browse and select the "bin" subfolder of where you installed/extracted your ARM toolchain
4. Click Apply and OK

OpenOCD

OpenOCD is used as a debugging interface for embedded platforms. It forms the connection between GDB and the hardware. The tested version is 0.10.0-12.

The SWD adapter for programming and debugging is integrated on the Mainboard from v11 onwards and on the Base Station since v3. Only a micro USB cable is needed.

Installation

1. Pre-built and up-to-date binaries for Windows, Linux (Debian-based), and OSX can be found [here](#).
2. For windows:
 1. Unpack the content to a folder of your choice
 2. To use the debugging hardware the WinUSB driver is required
 3. This is most easily installed with Zadig's tool found here: <http://zadig.akeo.ie/>, download it and start it while your adapter is plugged in
 4. Select: Options => List All Devices
 5. In the drop-down list select the correct device. E.g. "Mainboard v11 (Interface 0)". For the mainboard and base station always select "Interface 0".
 6. Select the WinUSB driver and click "Replace driver", confirm the installation
 7. Close the tool, Done!
3. For Linux:
 1. Download a prebuild binary from GNU MCU: <https://github.com/gnu-mcu-eclipse/openocd/releases>
 2. Alternative: You can check your package manager, but it will most likely be incompatible (really, no need to give it a try...)
 3. Alternative: You can build it yourself with below instructions (reference: http://www.elinux.org/Compiling_OpenOCD_v06_Linux)
 1. Here is the short version (but you may need additional libs from package manager):

```
git clone git://git.code.sf.net/p/openocd/code openocd
cd openocd
./bootstrap
./configure --enable-ftdi
make
sudo make install
```

2. The binary will be installed to /usr/local/bin/, you can use this path later on in this HowTo.

4. To run OpenOCD you usually require super user rights. Instructions on how to avoid this can be found here: http://elinux.org/Accessing_Devices_without_Sudo

Configure OpenOCD

1. Open: Window => Preferences
2. Go to: MCU => Global OpenOCD Path
3. Set the executable to "openocd.exe" (on Windows) or "openocd" (on Linux).
4. Set the folder variable to the "bin" subfolder of your OpenOCD installation.
5. Click OK and close the window

Project Setup and Usage

The firmware project contains all the code for our base station (BS) and for our robot (MB). There are different build configurations to select what to build.

For each processor there is one folder below the "app" folder and a separate project. The base station has only one processor, called "bs2018". The robot mainboard uses another microcontroller which is called "main2016".

For v2019 robots there is "main2019" for the primary microcontroller, "ir" for the infrared barrier processor and "motor" for motor controllers.

A separate project for each processor is required to make the Eclipse Indexer as happy as possible. Otherwise it would complain about multiple defines in the different processor include paths.

Furthermore, each processor has a custom bootloader. Therefore, the flash programming area has been divided into two parts. This allows the robot to be reprogrammed via our wireless interface. Although the base station also has a bootloader, this feature is currently not used.

Import from Archive

If you are using our public software release you should have a .zip archive containing the project.

1. In Eclipse select: File => Import...
2. Under "General" select "Existing Project Into Workspace", hit Next
3. Choose "Select archive file" and browse to the downloaded archive
4. In the "Projects:" box deselect all projects except the one labeled "Firmware"
5. Click finish
6. Once again, in Eclipse select: File => Import...
7. Under "General" select "Existing Project Into Workspace", hit Next
8. Choose "Select root directory" and browse to the location of the Firmware project on your disk
9. Check "Search for nested projects" and uncheck "Copy projects into Workspace"
10. Select all nested projects (bs2018, main2016, main2019, ir, motor) and click Finish

Import from GIT

If you are a member of the TIGERs team you can import the latest version from our GIT repository.

1. In Eclipse select: File => Import...
2. Under "GIT" select "Projects from Git", hit Next
3. Select "Clone URI", hit Next
4. The URI is: <https://gitlab.tigers-mannheim.de/main/Firmware.git>
5. Enter your username and password under "Authentication", click Next
6. Select at least the master branch, others are optional, click Next
7. Select a local directory for the project, click Next
8. Wait for the download to finish and then select "Import existing projects", click Next
9. Select all projects and click "Finish"

Compiling

1. Right click on the Firmware project and go to: Build Configurations => Set Active
2. There are six configurations, for the v2016 mainboard (MB2016), v2019 mainboard (MB2019) and for the Base Station (BS2018). Both in a debug and a release build.
3. Choose the one you wish to build
4. Right click on the project and select Build Project
5. You can also select the build configuration and the build command in the toolbar. It is the small hammer symbol and the symbol left of it. Make sure you select the Firmware project before using the buttons.
6. Building the project via the build command always compiles the bootloader and the run code. You may select a more fine grained control by using the configured make targets. If the "Make Target" view is not open, select it via Window => Show View. The make targets can be found in the Firmware folder.

Flashing

1. Flashing of all processors is possible via a "Run Configuration"
2. Go to: Run => Run Configurations...
3. There are entries for flashing the base station or the mainboard (each with bootloader and run configuration)
4. Just select the desired configuration and hit Run (requires OpenOCD to be setup correctly)
5. If there is no code on the processors you will need to flash the bootloader first
6. This can also be selected in the toolbar (white arrow in green circle)

Debugging

Each processor can be debugged individually

1. Go to: Run => Debug Configurations...
2. You can find all configurations already set up under the "GDB OpenOCD Debugging" node
3. Select the processor and mode (bootloader/run) you wish to debug and click "Debug" (requires OpenOCD to be setup correctly)
4. The appropriate debug build image will automatically be loaded onto the corresponding processor
5. Happy debugging!