POSIX programming on the EV3 robot

1 Toolchain configuration

It takes a while, I know, but it is a necessary phase for most embedded systems development. It took me around 4 full work days of tries and errors to figure out a way to make it all work, while I have experience in toolchain configuration. This first part is a whole part of the labs, please read thoroughly the explanations given for each step, they are meant to transmit a part of this experience to you.

1.1 Eclipse installation

Just extract the eclipse neon zip file wherever you want

1.2 ARM cross compiler installation

Install the CodeSourcery executable file, downloaded from :

https://sourcery.mentor.com/GNUToolchain/package4574/public/arm-none-linux-gnueabi/arm-2009q1-203-arm-none-linux-gnueabi.exe

If you do not have admin rights, choose custom install and install it in a folder where you have writing perms.

1.3 C wrapper for EV3

Copy the ev3dev-c folder wherever you want, just remember where it is. It is a modified version of the original https://github.com/in4lio/ev3dev-c project. My modifications concerned the Makefile of the source/ev3 folder, where basically I changed the options such that the library, while compiled on a host computer, acts as if it were compiled on the EV3 robot, in order to create the static library lib/ev3dev-c.a that will be used for cross compiling a program that will be executed on the EV3.

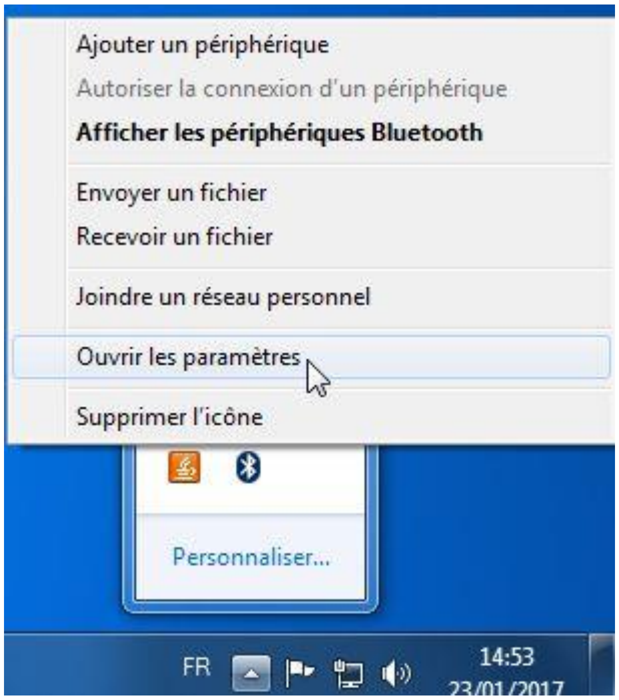
2 Installation of ev3dev on the EV3 (it is already done on our EV3 robots)

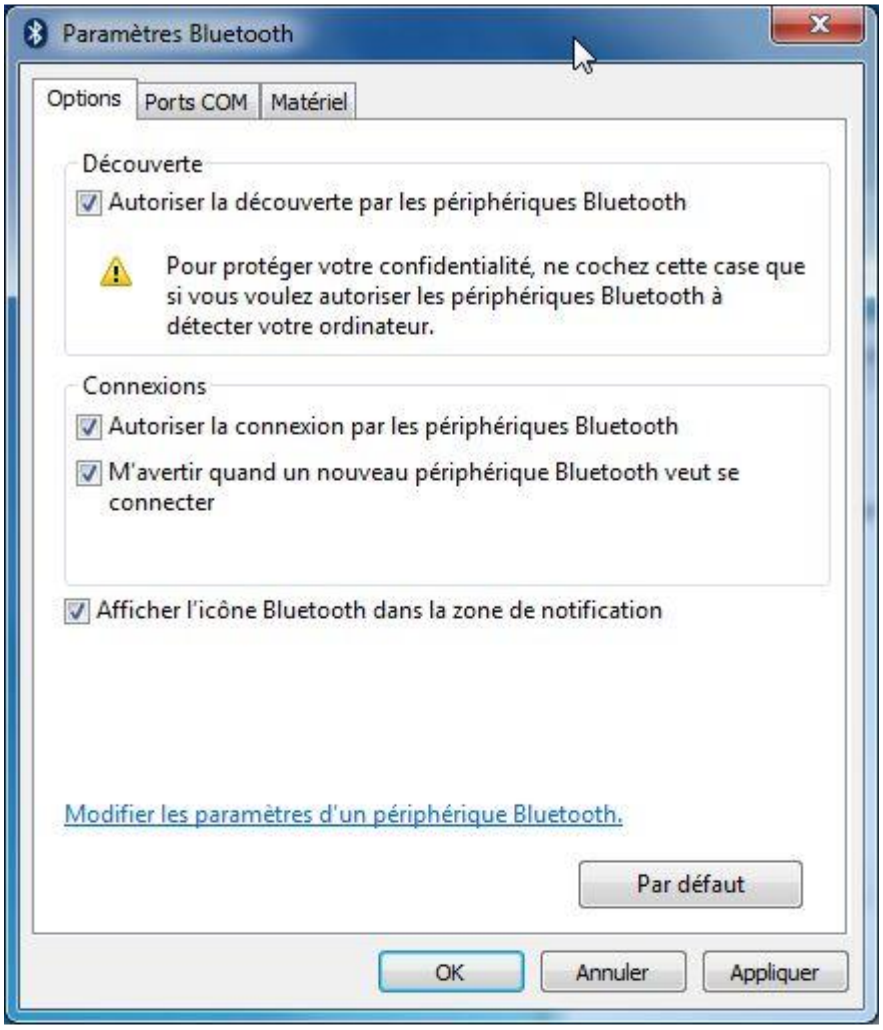
This step is already done on the EV3 robots, so you do not have to flash the cards. For information only, the ev3dev starting page is explaining the steps 1 to 4 to create a bootable SD card such that the EV3 will boot on it: http://www.ev3dev.org/docs/getting-started/.

3 Bluetooth tethering

3.1 Bluetooth association

If the host computer does not have internal Bluetooth, first you need to plug in the Bluetooth USB key, and wait for the drivers to install themselves. Then you should see a Bluetooth icon in the status bar. Make the host computer visible on Bluetooth:





Découverte

[✓] Autoriser la découverte par les périphériques Bluetooth

Pour protéger votre confidentialité, ne cochez cette case que si vous voulez autoriser les périphériques Bluetooth à détecter votre ordinateur.

Connexions

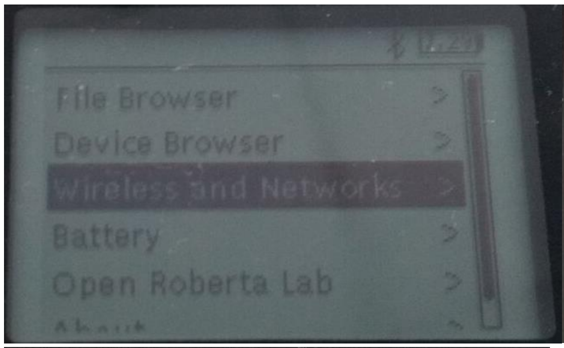
[✓] Autoriser la connexion par les périphériques Bluetooth

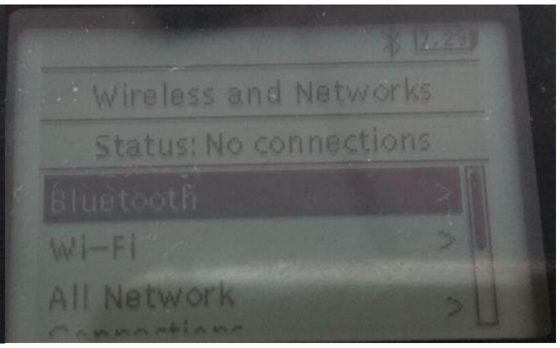
[✓] M'avertir quand un nouveau périphérique Bluetooth veut se connecter

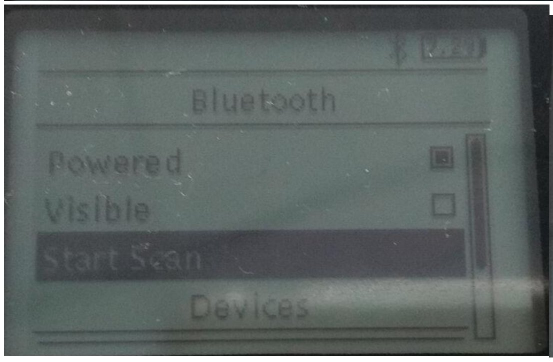
[✓] Afficher l'icône Bluetooth dans la zone de notification

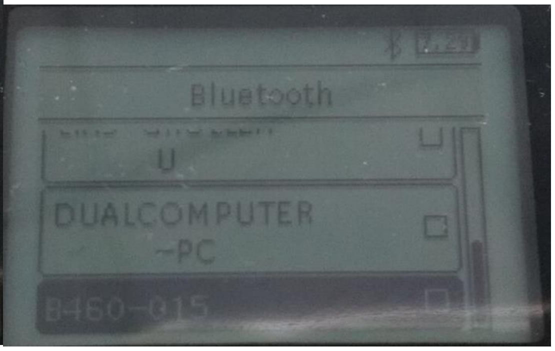
Modifier les paramètres d'un périphérique Bluetooth.

Then go to the EV3 robot and start scanning for Bluetooth devices until you can see the host computer, on our example B460-015.

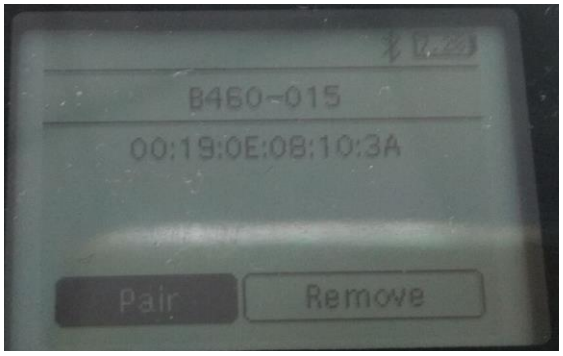




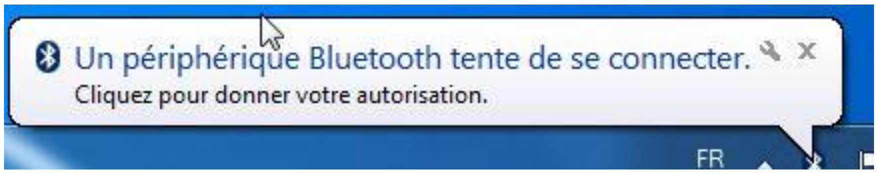




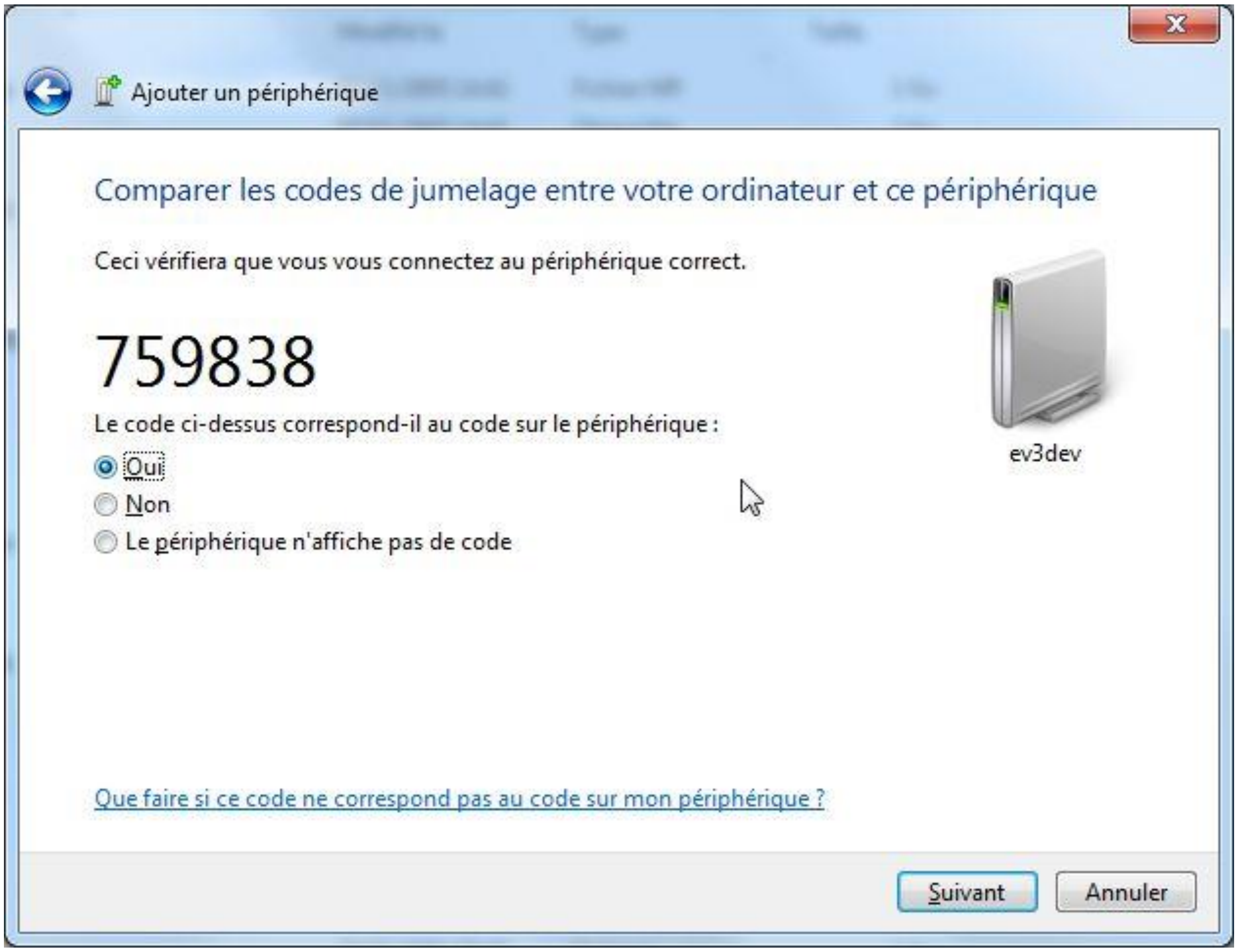
Select this device, and Pair it.



Be quick to respond on the host computer, by clicking on the notification:


Un périphérique Bluetooth tente de se connecter.
Cliquez pour donner votre autorisation.

Then validate the default pairing code on the host computer as well as on the EV3:


Ajouter un périphérique

Comparer les codes de jumelage entre votre ordinateur et ce périphérique

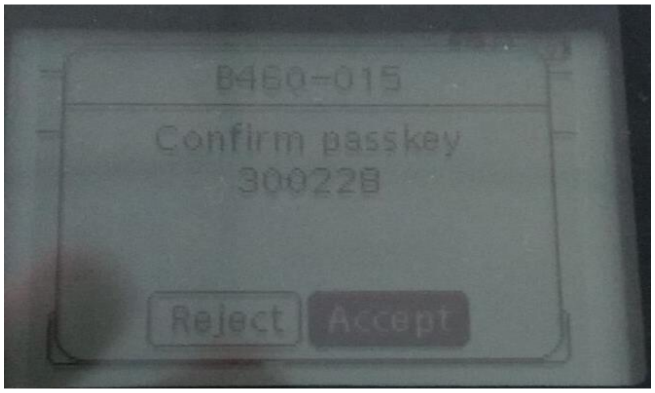
Ceci vérifiera que vous vous connectez au périphérique correct.

759838

Le code ci-dessus correspond-il au code sur le périphérique :

[✓ Oui]
Non
Le périphérique n'affiche pas de code

ev3dev
Que faire si ce code ne correspond pas au code sur mon périphérique ?
Suivant
Annuler


B460-015

Confirm passkey
300228

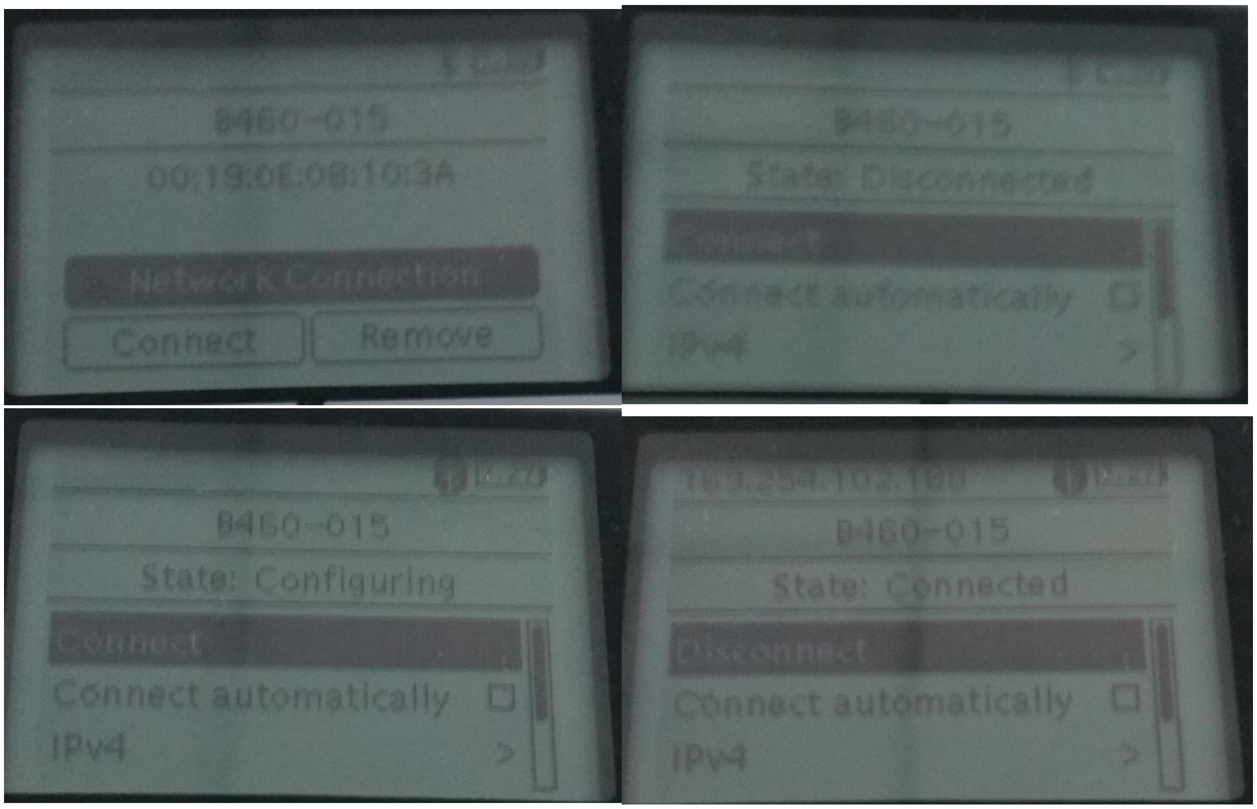
Note: the EV3 picture and the screenshot passcode do not match, because I was never quick enough to take a picture of the EV3 screen while taking a screenshot… Normally the passkeys match of course.

Now the host computer and the EV3 are paired.

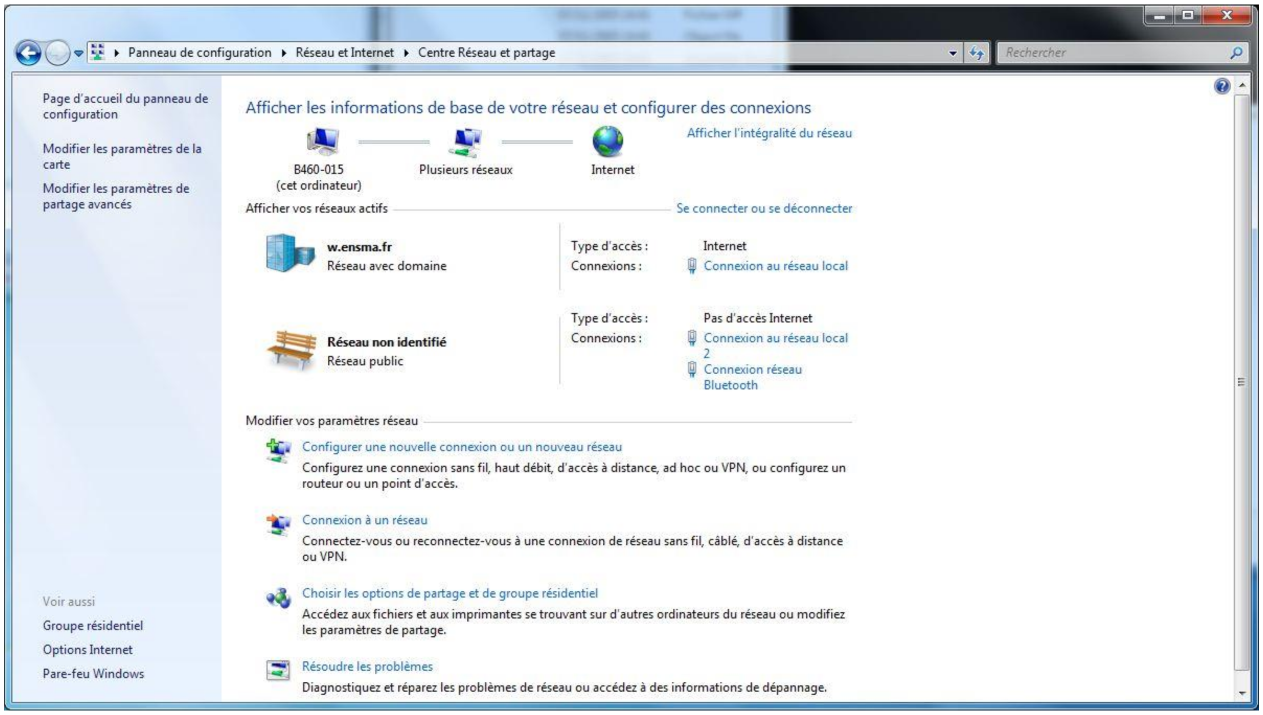
Troubleshooting: pairing Bluetooth devices may be tricky. For example, when one of the devices think it is paired to the other one, but not the other, it might fail. On my host computer, removing a Bluetooth device from the paired ones is failing most of the time. A good workaround is to delete the host Bluetooth controller from the device control panel, and then to let Windows reinstall it.

3.2 Tethering

We now need to create a personal area network, such that IP is possible over Bluetooth. On the EV3, press on “Network Connection”, then “Connect”, and wait, it can take up to a minute of configuration, and even raise a timeout error (which is not a problem).



When the connection is established, the EV3 has an IP address, on the picture it is 169.254.102.188 (we will call it EV3IP in the sequel). This IP address is part of a personal area network (PAN), we can see this PAN in the network control panel of the host computer:


Panneau de configuration ▸ Réseau et Internet ▸ Centre Réseau et partage

Page d'accueil du panneau de configuration
Modifier les paramètres de la carte
Modifier les paramètres de partage avancés

Afficher les informations de base de votre réseau et configurer des connexions

B460-015 (cet ordinateur) — Plusieurs réseaux — Internet
Afficher l'intégralité du réseau

Afficher vos réseaux actifs — Se connecter ou se déconnecter

w.ensma.fr
Réseau avec domaine
Type d'accès : Internet
Connexions : Connexion au réseau local

Réseau non identifié
Réseau public
Type d'accès : Pas d'accès Internet
Connexions : Connexion au réseau local 2
Connexion réseau Bluetooth

Modifier vos paramètres réseau

Configurer une nouvelle connexion ou un nouveau réseau
Configurez une connexion sans fil, haut débit, d'accès à distance, ad hoc ou VPN, ou configurez un routeur ou un point d'accès.

Connexion à un réseau
Connectez-vous ou reconnectez-vous à une connexion de réseau sans fil, câblé, d'accès à distance ou VPN.

Choisir les options de partage et de groupe résidentiel
Accédez aux fichiers et aux imprimantes se trouvant sur d'autres ordinateurs du réseau ou modifiez les paramètres de partage.

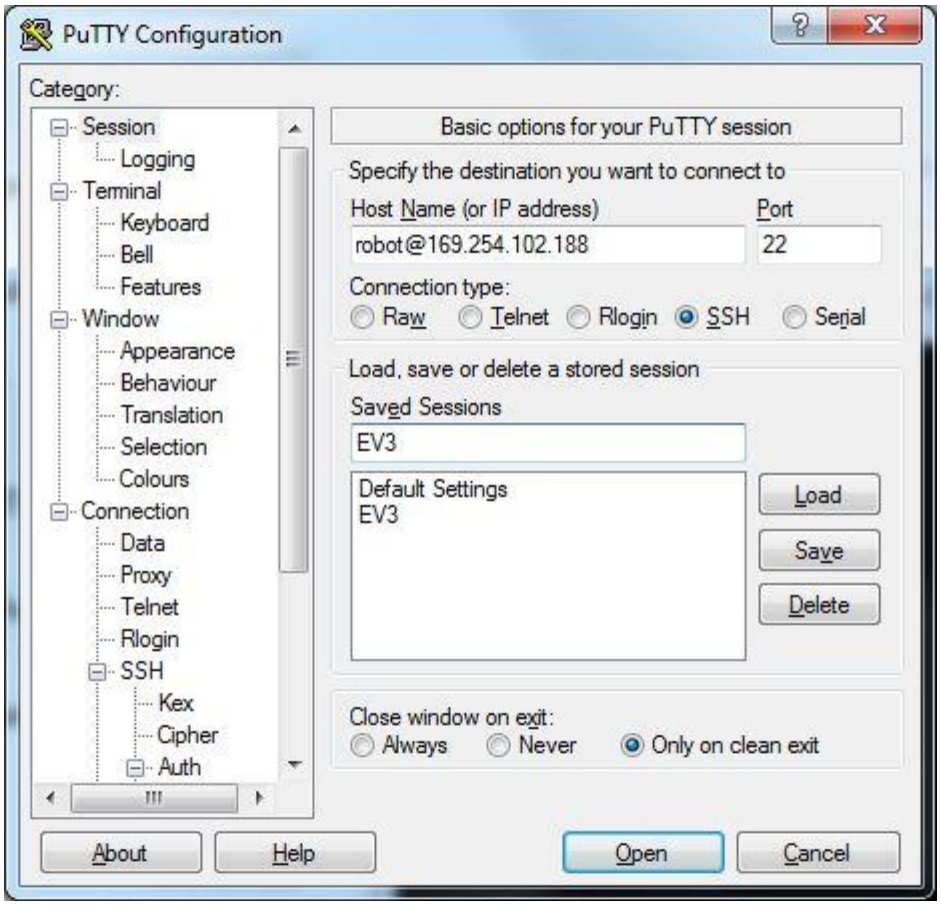
Résoudre les problèmes
Diagnostiquez et réparez les problèmes de réseau ou accédez à des informations de dépannage.

Voir aussi
Groupe résidentiel
Options Internet
Pare-feu Windows

There we go, now the EV3 and our computer are able to communicate over IP.

3.3 Testing

On the host, extract the putty.zip file in a folder, then run putty.exe. Create a SSH session, using robot@EV3IP. When opening the session for the first time, we will get a security alert because a new SSH key is generated, just validate.


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): robot@169.254.102.188

Port: 22

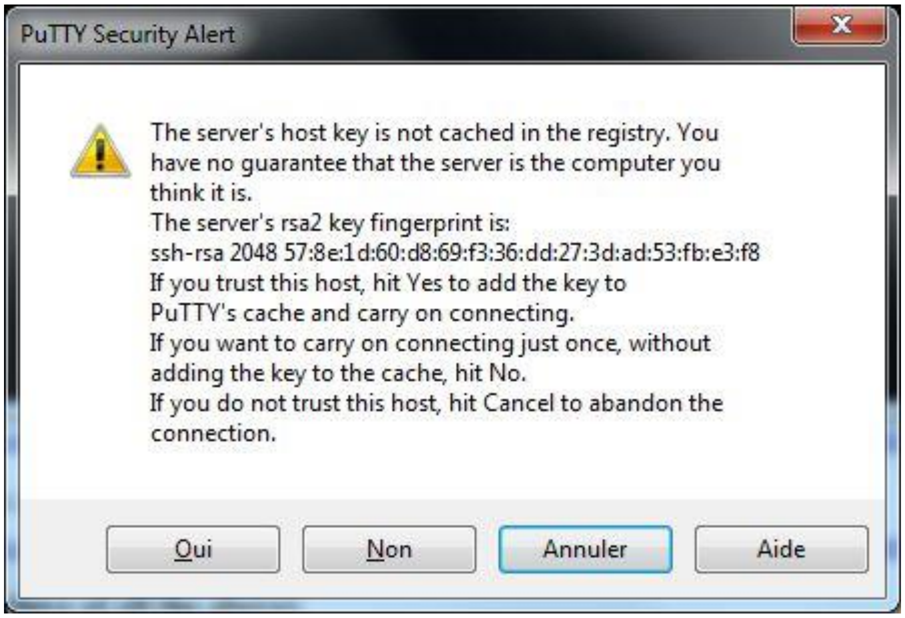
Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions: EV3

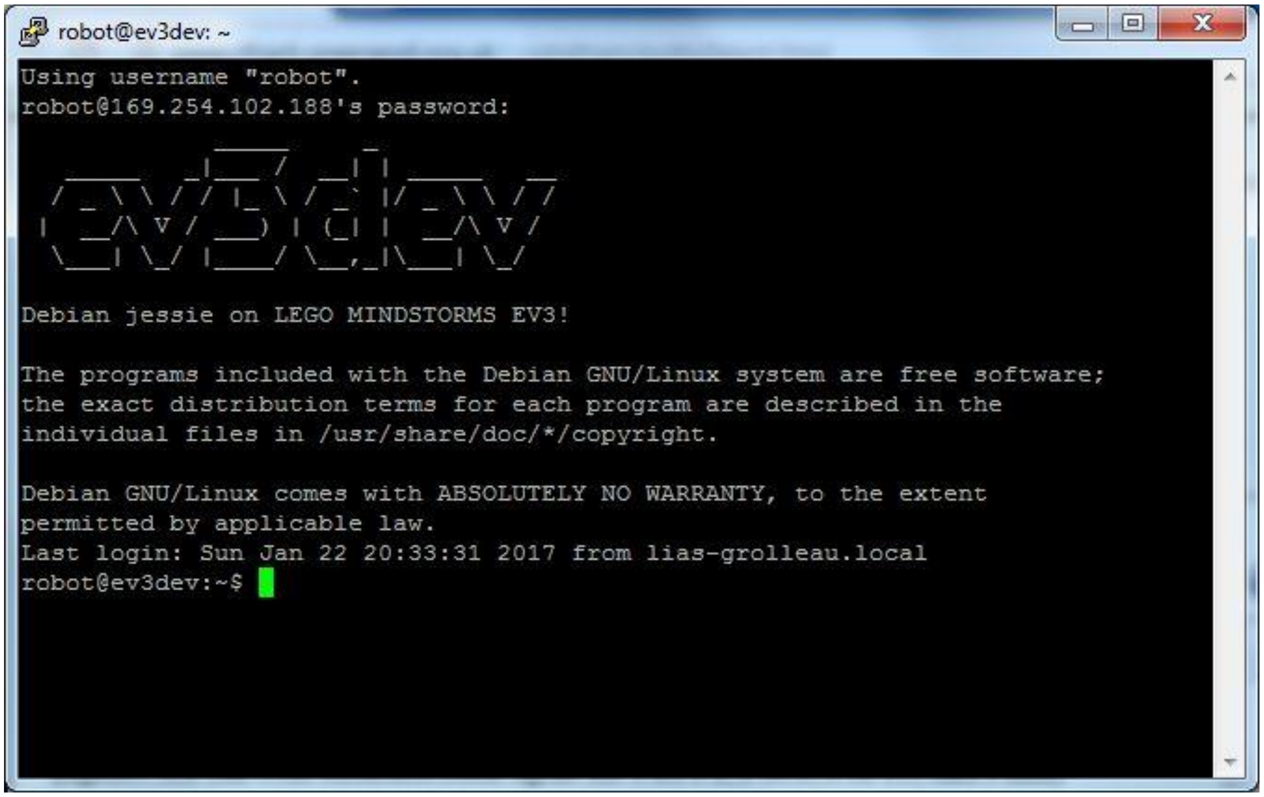
Default Settings
EV3

Close window on exit: Always Never [✓ Only on clean exit]


PuTTY Security Alert

The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 2048 57:8e:1d:60:d8:69:f3:36:dd:27:3d:ad:53:fb:e3:f8
If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the connection.

The default password of ev3dev is `maker`. Here we go: a SSH shell is opened from the host computer to the ev3 over IP.



robot@ev3dev: ~

```
Using username "robot".
robot@169.254.102.188's password:
             _____     _
   _____   _|___ /  __| | _____   __
  / _ \ \ / / |_ \ / _` |/ _ \ \ / /
 |  __/\ V / ___) | (_| |  __/\ V /
  \___| \_/ |____/ \__,_|\___| \_/

Debian jessie on LEGO MINDSTORMS EV3!

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jan 22 20:33:31 2017 from lias-grolleau.local
robot@ev3dev:~$
```

Keep this SSH window opened, it may be useful later to kill our program if it does not stop.

3.4 Playing with the shell

Ev3dev is a Linux and a lot of drivers for the EV3 sensors/actuators. For example, in order to control the tacho motors, we will find in the folder /sys/class/tacho-motor one folder per tacho motor connected to the ev3. Let's see what happens when playing with the 3rd motor (motor2):

```
robot@ev3dev:/sys/class/tacho-motor/motor2$ ls
address         driver_name     polarity       ramp_up_sp   stop_action
command         duty_cycle      position       speed        stop_actions
commands        duty_cycle_sp   position_sp    speed_pid    subsystem
count_per_rot   hold_pid        power          speed_sp     time_sp
device          max_speed       ramp_down_sp   state        uevent
```

These files are used for reading (example, `cat count_per_rot` displays the amount of tachometer per rotation, here 360) or writing, for example `echo stop > command` will stop the motor. Think about that if ever your ev3 motors are running while your program is not controlling them anymore (crash, killed, etc.). For more information on what everything does, consult the hardware drivers documentation on the ev3dev project website http://www.ev3dev.org/docs/driver-overview/. For example, below we set the duty_cycle_sp property to tell the motor to go to 100% power when we will ask it to move. Then we give an absolute position objective of 45°, we ask the tacho motor to move to this absolute position, and do the same for the position 0°. When doing this kind of action, it is good to tell the motor to hold the position on stop. In this mode, we will not even be able to move the motor by hand. If we want to be able to move it by hand, we will need to change the stop action to `coast`.

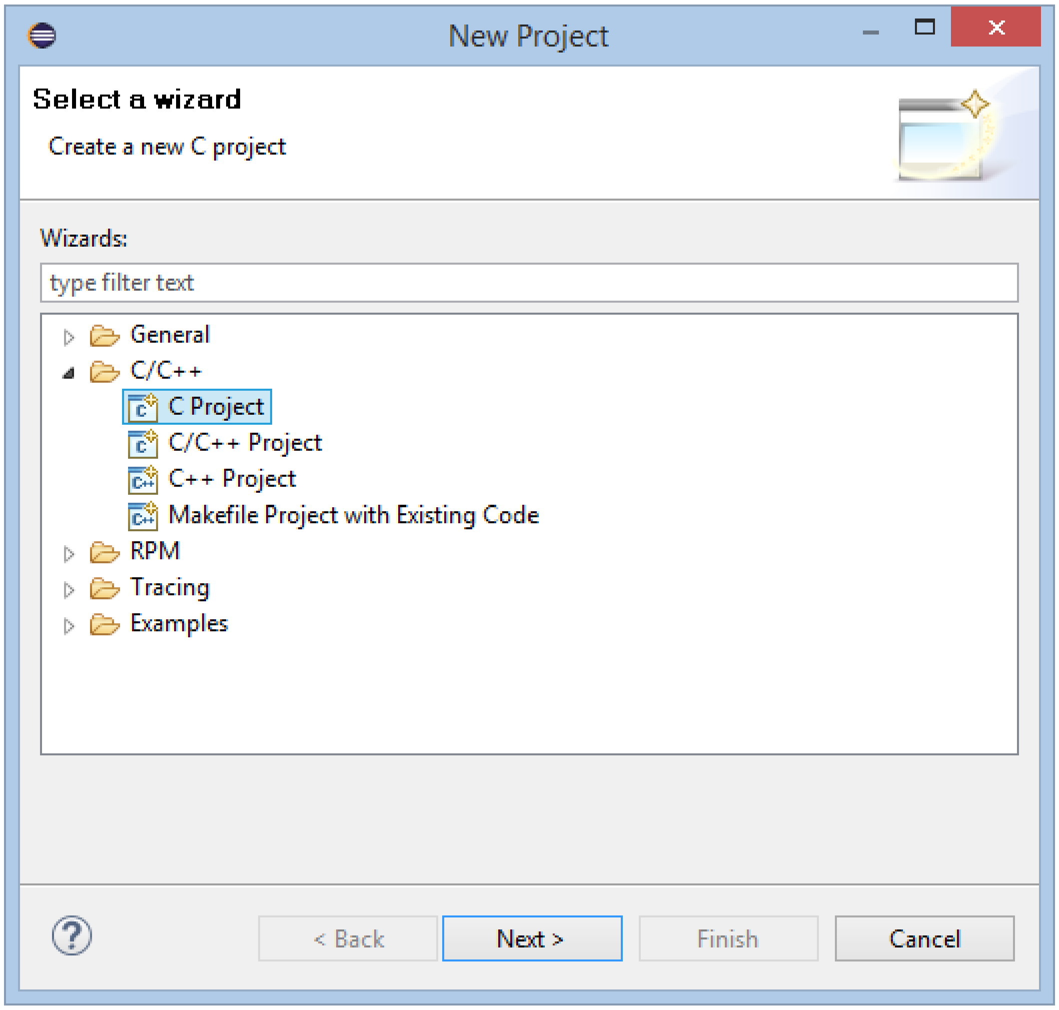
```
robot@ev3dev:/sys/class/tacho-motor/motor2$ echo 100 > duty_cycle_sp
robot@ev3dev:/sys/class/tacho-motor/motor2$ cat duty_cycle_sp
100
robot@ev3dev:/sys/class/tacho-motor/motor2$ cat position_sp
0
robot@ev3dev:/sys/class/tacho-motor/motor2$ echo 45 > position_sp
robot@ev3dev:/sys/class/tacho-motor/motor2$ echo run-to-abs-pos > command
robot@ev3dev:/sys/class/tacho-motor/motor2$ echo 0 > position_sp
robot@ev3dev:/sys/class/tacho-motor/motor2$ echo run-to-abs-pos > command
robot@ev3dev:/sys/class/tacho-motor/motor2$ cat stop_action
hold
robot@ev3dev:/sys/class/tacho-motor/motor2$ cat stop_actions
coast brake hold
robot@ev3dev:/sys/class/tacho-motor/motor2$ echo coast > stop_action
```

The ev3dev-c library that we will use to program the ev3 in C is just a C wrapper for all the drivers.

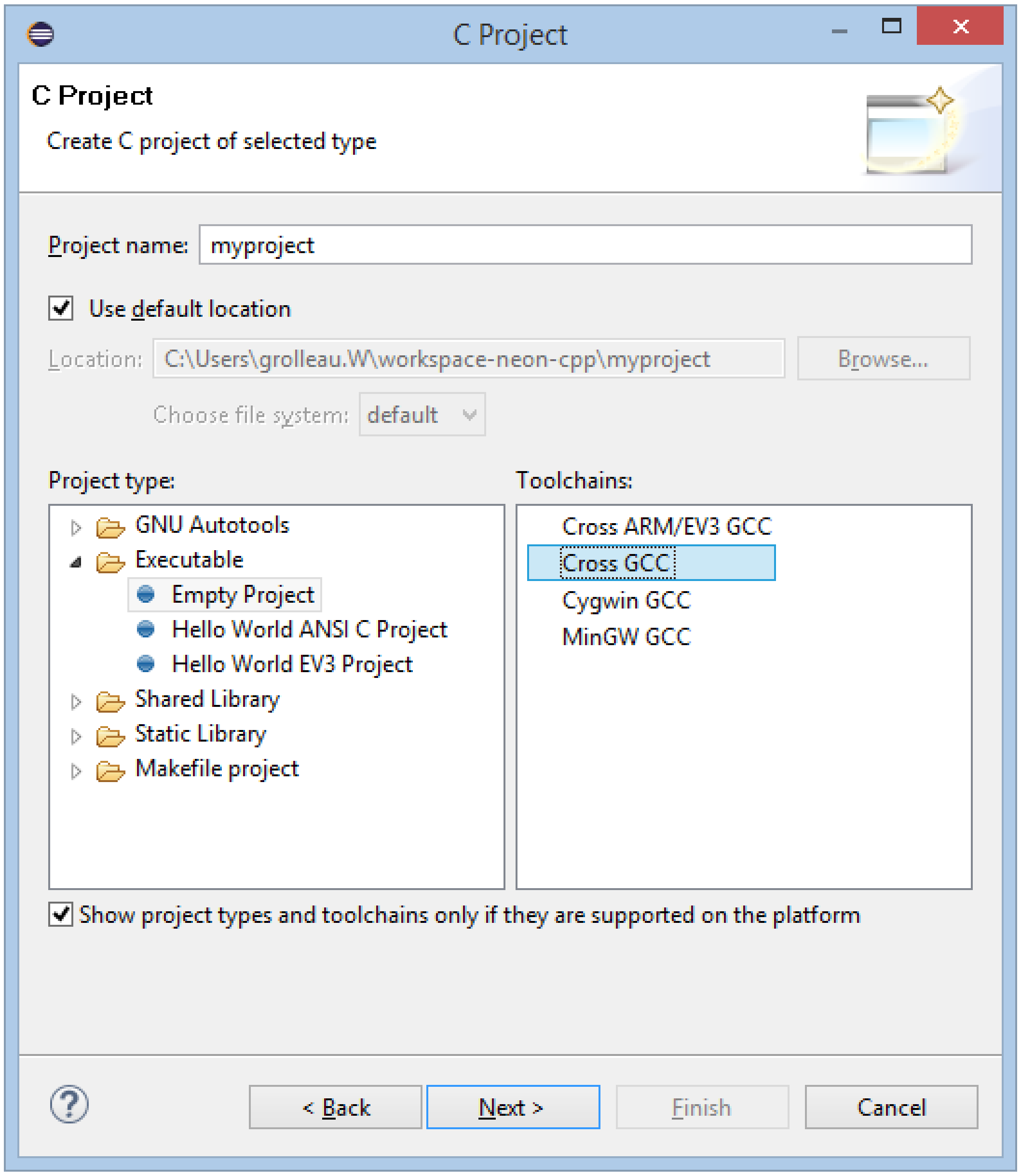
4 Create an eclipse project

4.1 Configure the project for building for ev3 running ev3dev

Run eclipse, pick your workspace folder. Then create a new C project:



Name your project and pick the cross GCC compiler:



C Project

C Project

Create C project of selected type

Project name: myproject

☑ Use default location

Location: C:\Users\grolleau.W\workspace-neon-cpp\myproject

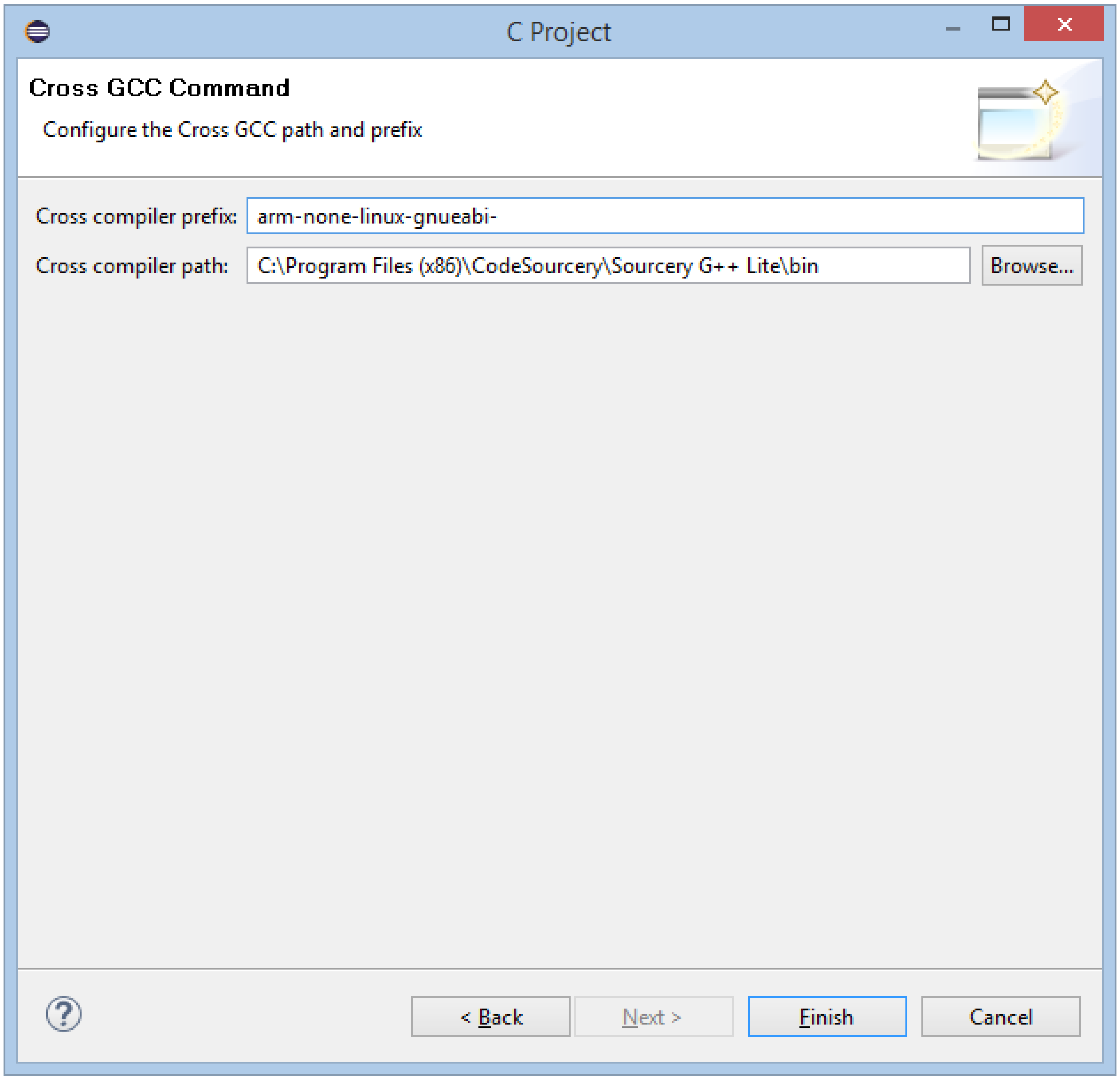
Choose file system: default

Project type:
- GNU Autotools
- Executable
  - Empty Project
  - Hello World ANSI C Project
  - Hello World EV3 Project
- Shared Library
- Static Library
- Makefile project

Toolchains:
- Cross ARM/EV3 GCC
- Cross GCC
- Cygwin GCC
- MinGW GCC

☑ Show project types and toolchains only if they are supported on the platform

For cross compiler prefix, enter `arm-none-linux-gnueabi-`, while for the cross compiler path, browse to the `bin` folder of CodeSourcery. This step is pretty straightforward, the GCC cross compiler command is nammed arm-none-linux-gnueabi-gcc, etc (see in the CodeSourcery bin folder):



Cross GCC Command

Configure the Cross GCC path and prefix

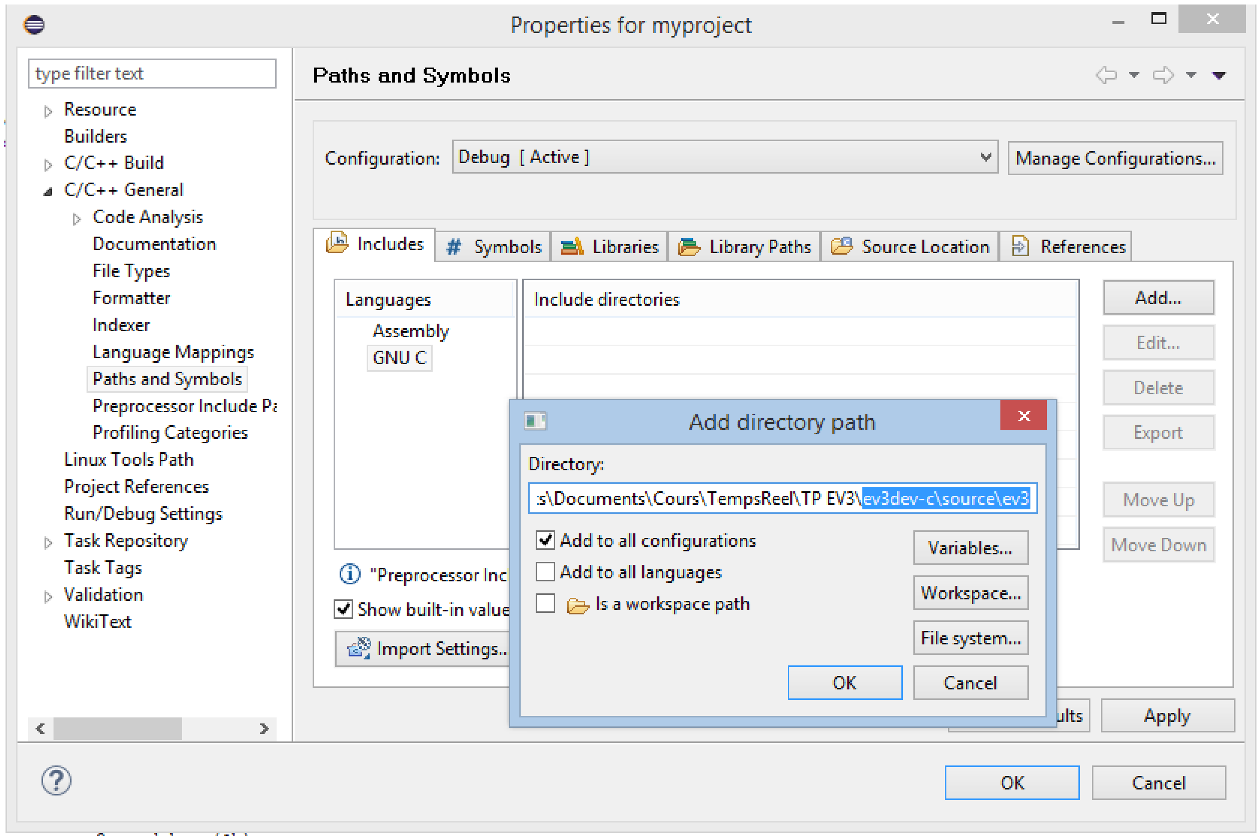
Cross compiler prefix: arm-none-linux-gnueabi-

Cross compiler path: C:\Program Files (x86)\CodeSourcery\Sourcery G++ Lite\bin

Now the most tricky part, the options. Right click on your project in the project explorer, and click on Properties. From the properties window, go to "C/C++ General" -> "Paths and Symbols".

In order to be able to write inclusions like `#include <ev3.h>`, we need to add the ev3dev-c source folder as an include folder. It will automatically be added with the option `-I` to the compile commands.

In the Includes tab, in the "Languages" part, click on "GNU C", click the button "Add…". Check "Add to all configurations", and browse "File system…" to the ev3dev-c/source/ev3 folder (which should be where you put it).



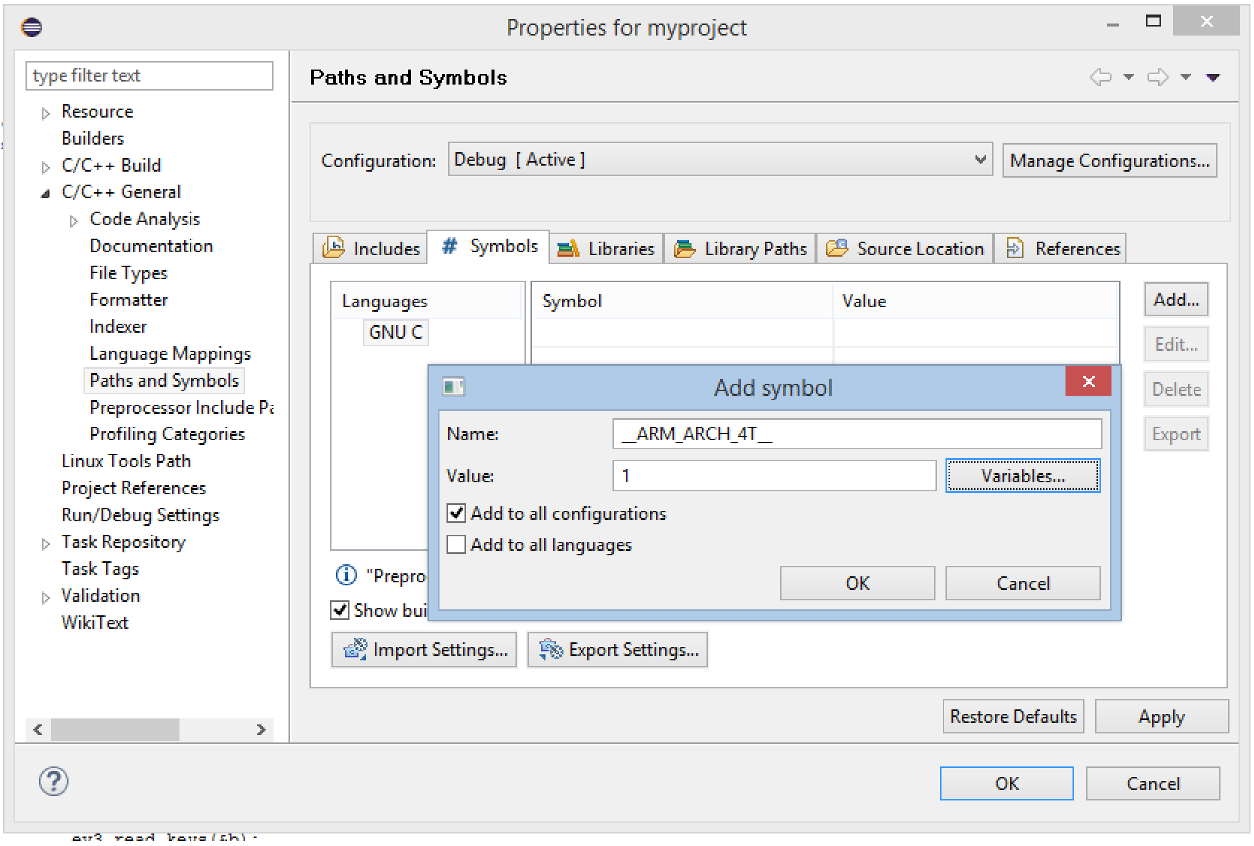
Configuration: Debug [Active]

Directory: :s\Documents\Cours\TempsReel\TP EV3\ev3dev-c\source\ev3

- [✓] Add to all configurations
- [ ] Add to all languages
- [ ] Is a workspace path
- [✓] Show built-in value

In order to trick ev3dev-c to think it is compiled on the ev3 robot, we need to define an environment variable that ev3dev-c is testing (with `#ifdef __ARM_ARCH_4T__`).

Go to “Symbols” tab, click on “Add…”, then enter __ARM_ARCH_4T__ (careful, there are two underline characters on each side) as Name, 1 as value, and check “Add to all configurations”:



Configuration: Debug [Active]

Name: __ARM_ARCH_4T__

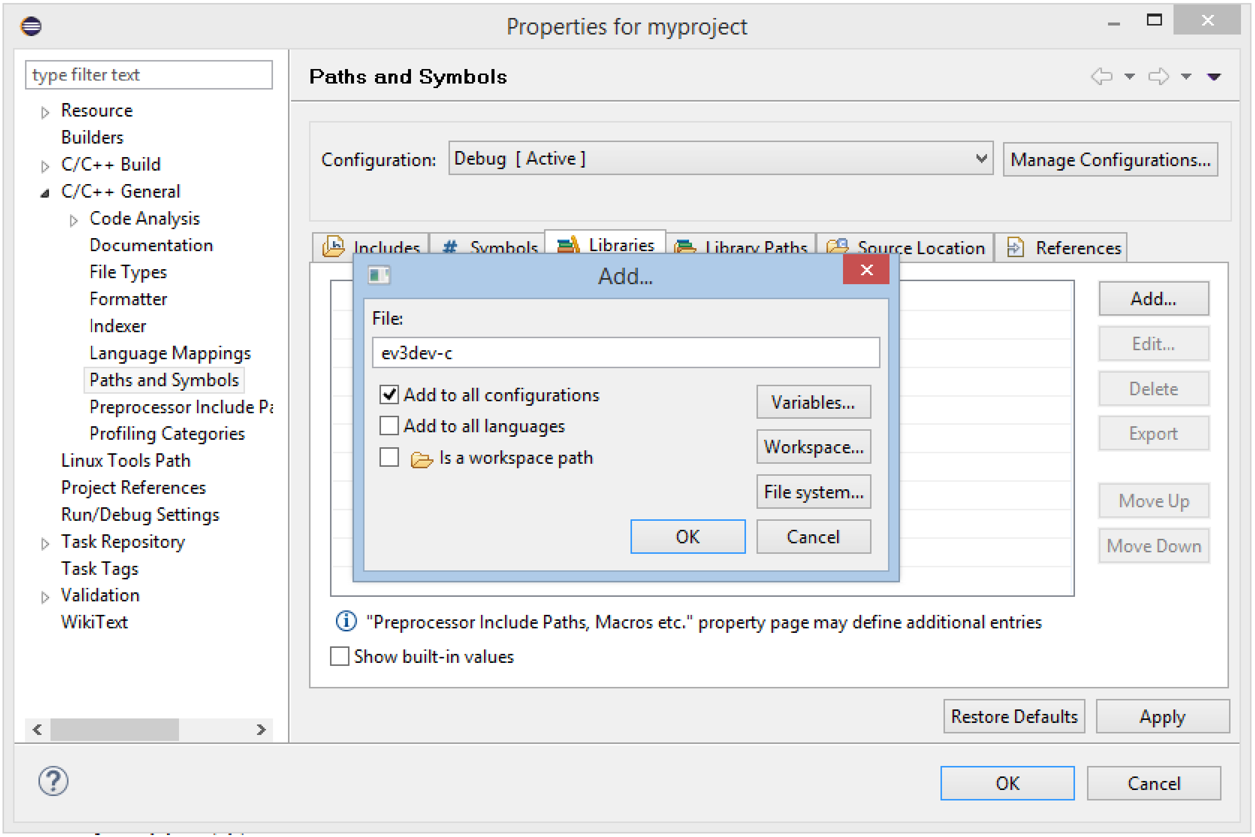
Value: 1

[✓] Add to all configurations

[ ] Add to all languages

When building the program, we will need to link to the "**ev3dev-c**" library (the file libev3dev-c.a in the lib folder of ev3dev-c). For gcc, we do not enter the `lib` prefix nor the `.a` extension. We will also need to add the math library "**m**" as well as the real-time library "**rt**". Doing so will automatically add to the linking stage `-lev3dev-c -lm -lrt`.

For this, we need to go to the next tab, the "Libraries" one. Click on "Add…", as usual, check "Add to all configurations", and enter `ev3dev-c` in the text box. Same for `m` and `rt` libraries.



Configuration: Debug [Active]

Add...

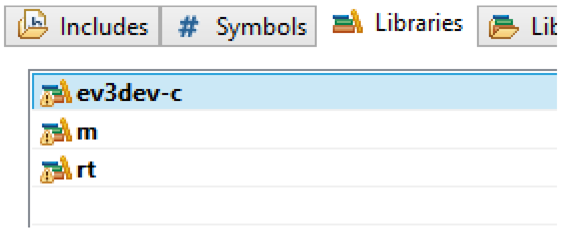
File: ev3dev-c

[✓] Add to all configurations
[ ] Add to all languages
[ ] Is a workspace path

"Preprocessor Include Paths, Macros etc." property page may define additional entries

[ ] Show built-in values

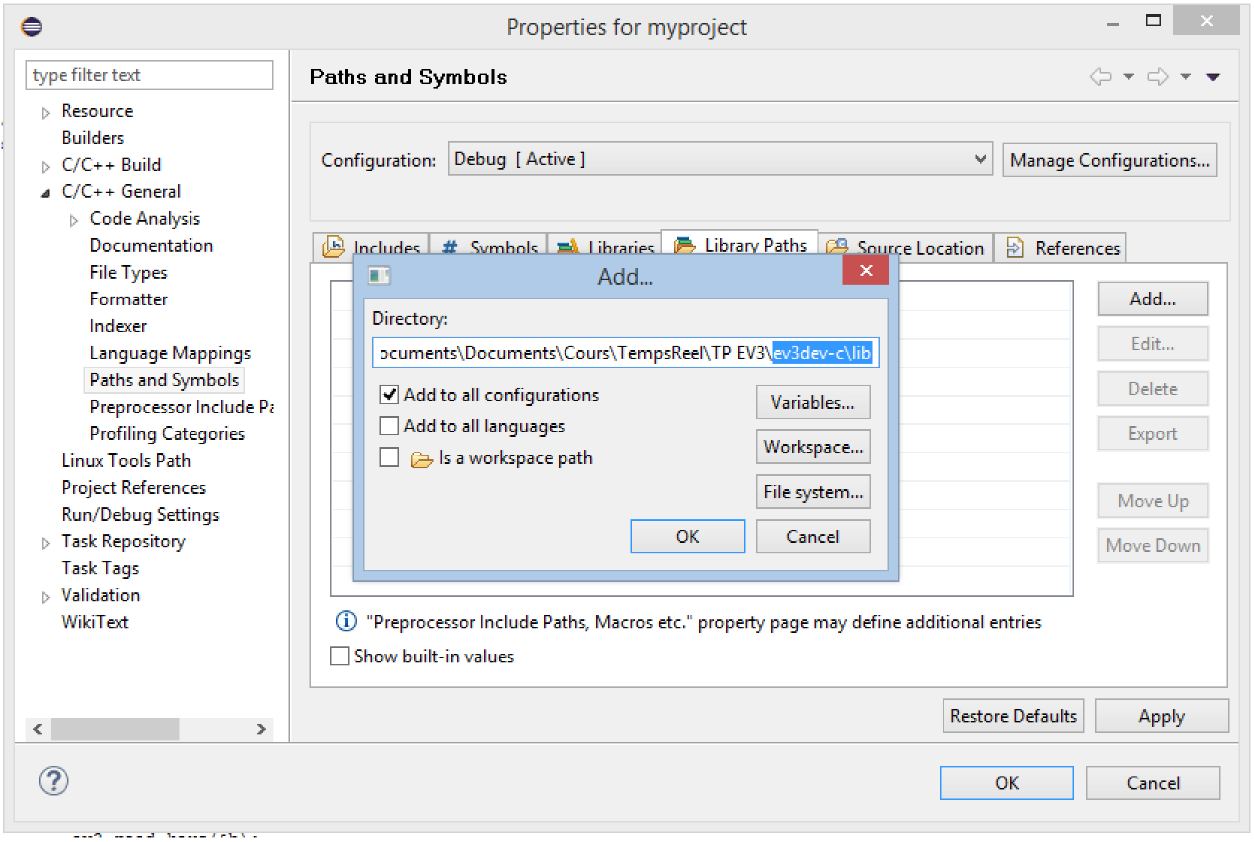
After adding these three libraries, you should see:



ev3dev-c
m
rt

The math and real-time libraries are located in the default library folders of the compiler. Nevertheless, ev3dev-c is located in a non standard folder, therefore, we need to add it.

In order to do so, we use the next tab, “Library paths”. Click on “Add…”, then as usual, check “Add to all configurations”, and browse to the ev3dev-c/lib folder (which should be, again, where you put it). It will automatically add the option –L followed by this folder to the linking stage.


Properties for myproject

Paths and Symbols

Configuration: Debug [Active]

Add...

Directory: ɔcuments\Documents\Cours\TempsReel\TP EV3\ev3dev-c\lib

[✓] Add to all configurations

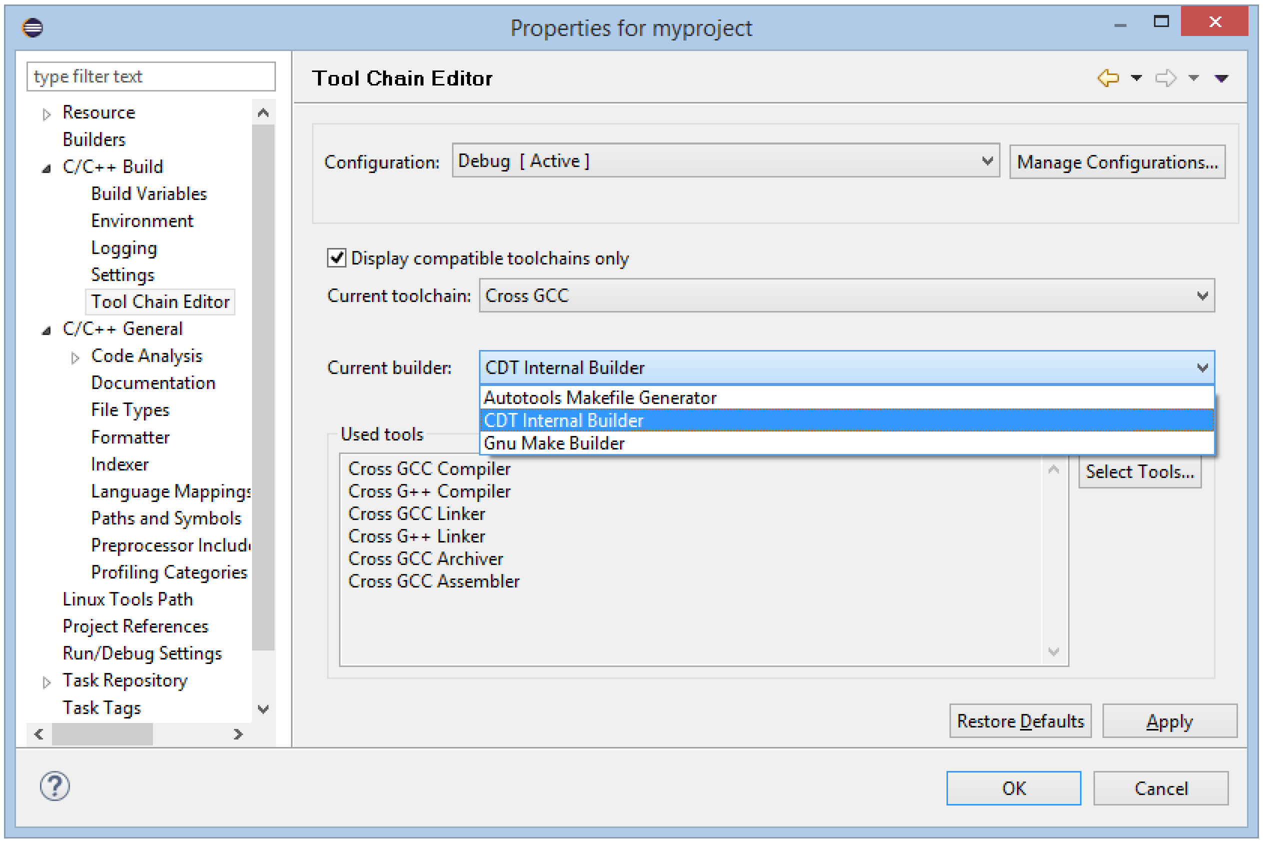
[ ] Add to all languages

[ ] Is a workspace path

"Preprocessor Include Paths, Macros etc." property page may define additional entries

[ ] Show built-in values

Last step, there is a bug in the default “Gnu make builder” (it creates multiple targets), so in the “Tool Chain Editor” property tab, change the “Current Builder” to “CDT Internal Builder”.


Properties for myproject

Tool Chain Editor

Configuration: Debug [Active]

☑ Display compatible toolchains only

Current toolchain: Cross GCC

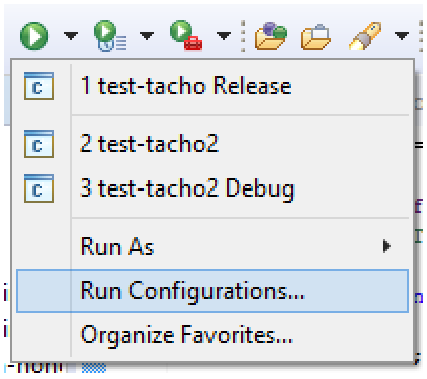
Current builder: CDT Internal Builder

Used tools:
Cross GCC Compiler
Cross G++ Compiler
Cross GCC Linker
Cross G++ Linker
Cross GCC Archiver
Cross GCC Assembler

To test it, create a sample C file with a main function doing a “hello world”, and build it.

4.2 Run configuration

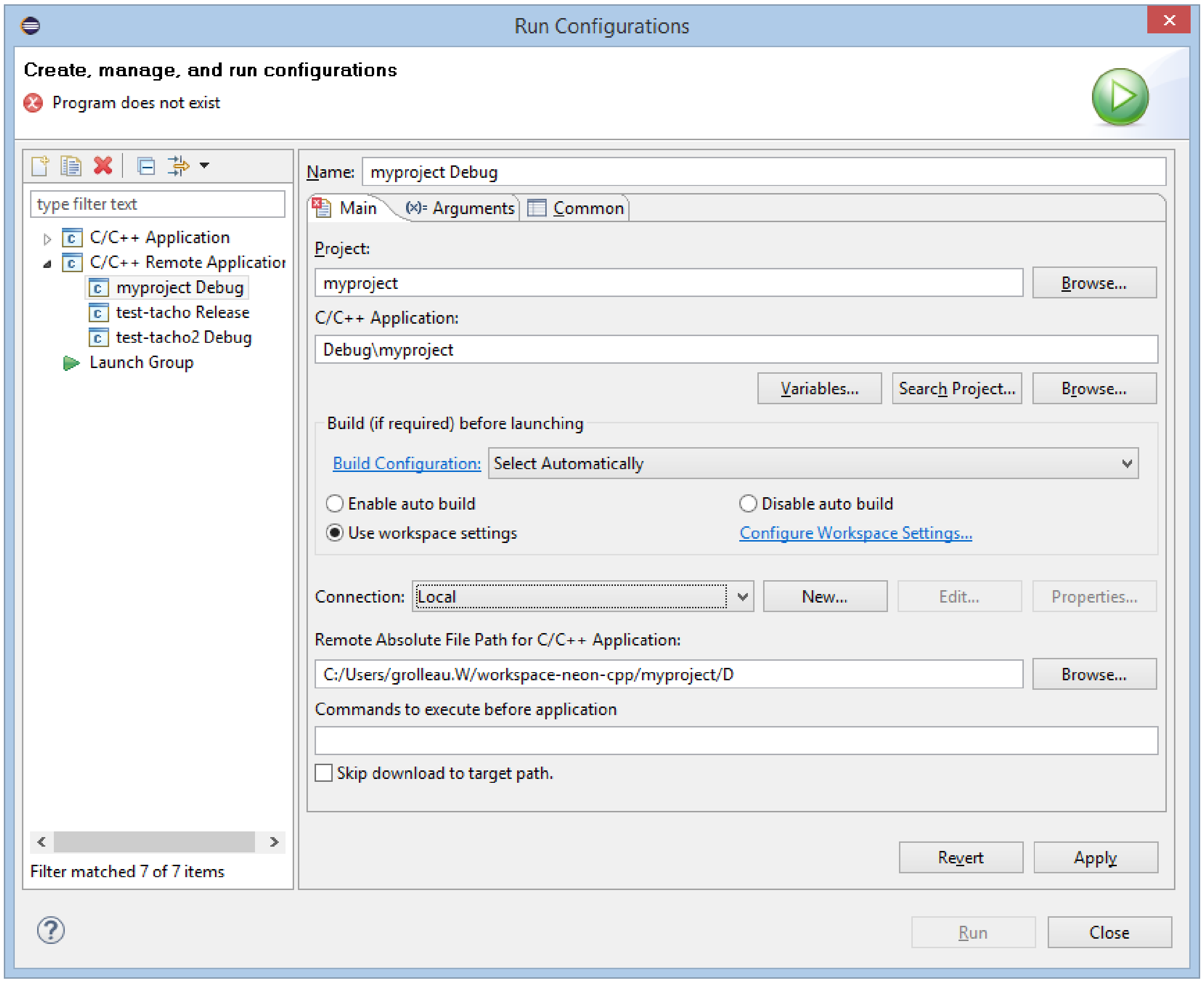
This last part will use the IP over Bluetooth tethering to upload and run the programs. First, we will create a running configuration:



The following steps will be done for Debug and Release configurations.

Let’s start with Debug configuration:

Create a "new C/C++ Remote Application", in the "C/C++ Application" text box, enter Debug\nameofproject. Careful, the field "Remote absolute file path for C/C++ Application" is the remote (i.e., on the robot) file path, it should be something like **/home/robot/myproject** (and not a "windows" path name like on the screenshot).


Run Configurations

Create, manage, and run configurations

Program does not exist

Name: myproject Debug

Project: myproject

C/C++ Application: Debug\myproject

Build (if required) before launching

Build Configuration: Select Automatically

Enable auto build
Disable auto build
● Use workspace settings
Configure Workspace Settings...

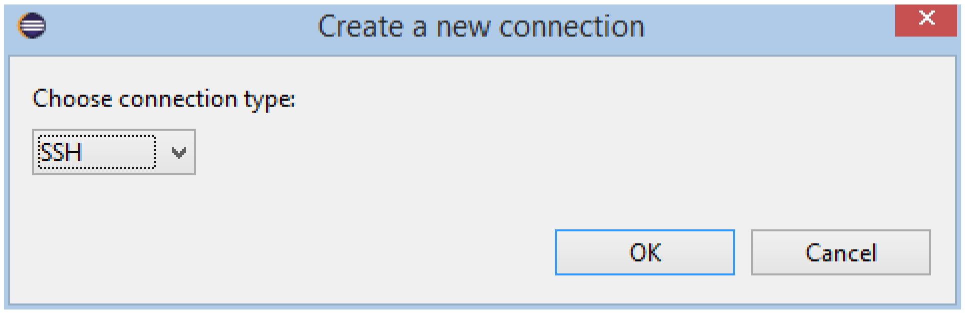
Connection: Local

Remote Absolute File Path for C/C++ Application: C:/Users/grolleau.W/workspace-neon-cpp/myproject/D

Commands to execute before application

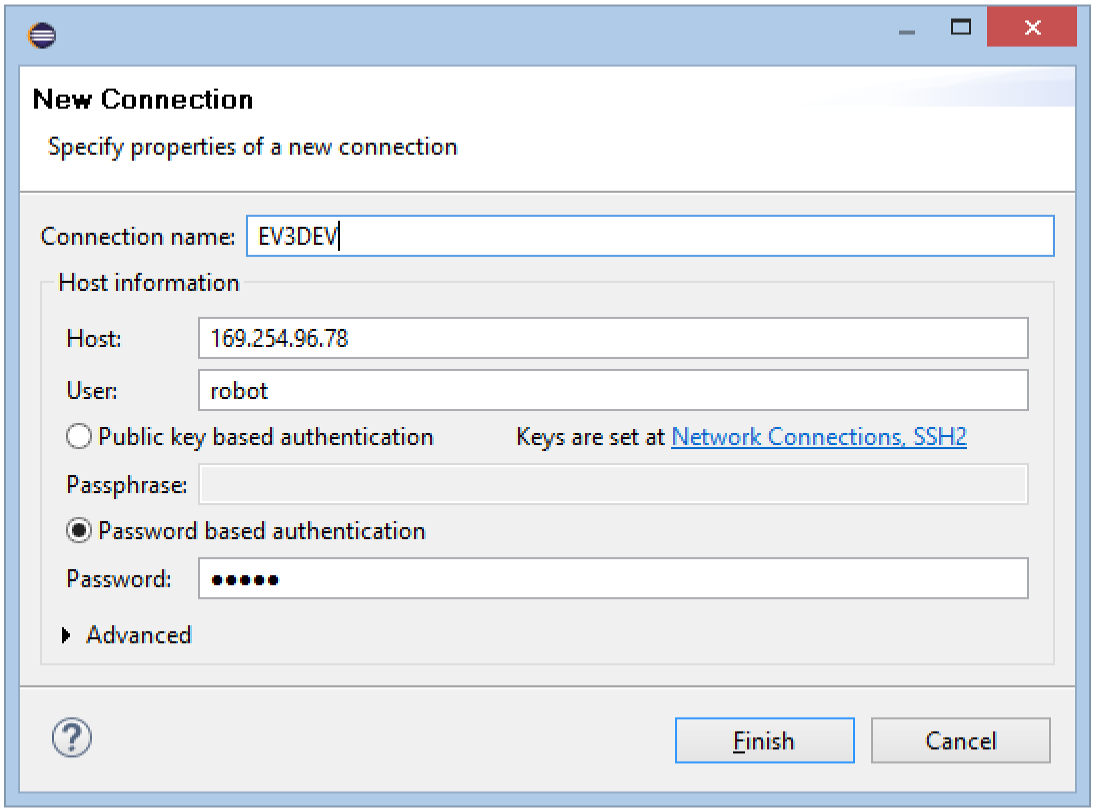
☐ Skip download to target path.

Close to "Connection", click on "New...", and select SSH.


Create a new connection

Choose connection type: SSH

Name it such that you remember it to be the EV3 robot. Enter the EV3 IP address in the "Host" text box, `robot` as user, check the radio button "Password based authentication" and enter `maker` as password.


New Connection

Specify properties of a new connection

Connection name: EV3DEV

Host information

Host: 169.254.96.78

User: robot

○ Public key based authentication   Keys are set at Network Connections, SSH2

Passphrase:

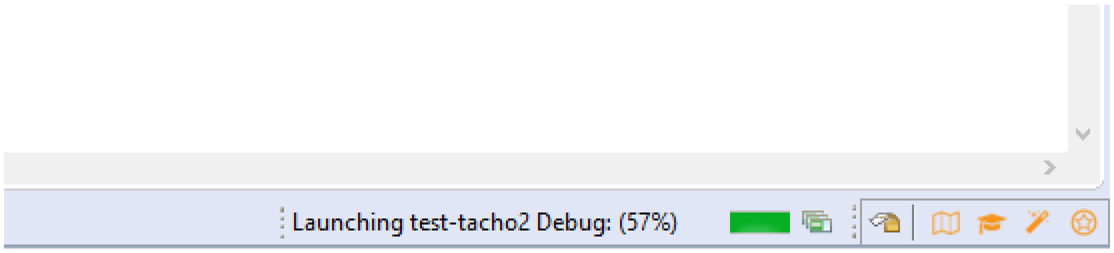
● Password based authentication

Password: ●●●●●

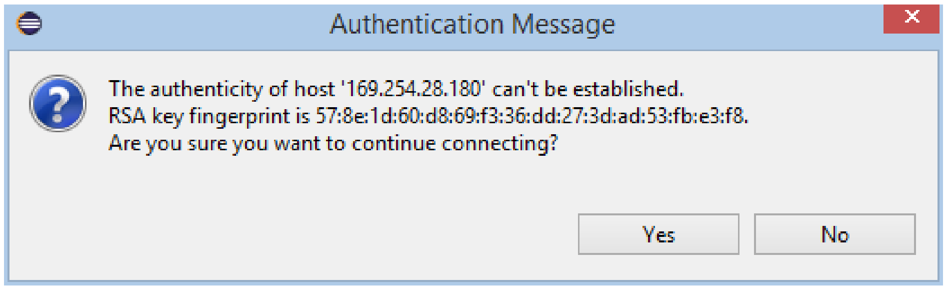
▸ Advanced

The first time we run this configuration may take a dozen of seconds, and we will have to validation the SSH validation code once and for all. After that it will take a couple of seconds. Using this running configuration, the program is built if necessary, then uploaded through Bluetooth tethering, and ran. Note that if our program is still running, and forbids us to upload the same program, we can always use the other SSH shell we opened to enter the command `pkill myproject`.

The first time we run the configuration, it may take a while, look at the bottom right of the eclipse window:


Launching test-tacho2 Debug: (57%)

You will also have to validate the SSH RSA key, like we did for putty:


Authentication Message

The authenticity of host '169.254.28.180' can't be established.
RSA key fingerprint is 57:8e:1d:60:d8:69:f3:36:dd:27:3d:ad:53:fb:e3:f8.
Are you sure you want to continue connecting?

5 The program

Import tp-ev3.zip as an archive eclipse project, it will be used as a basis for these labs.

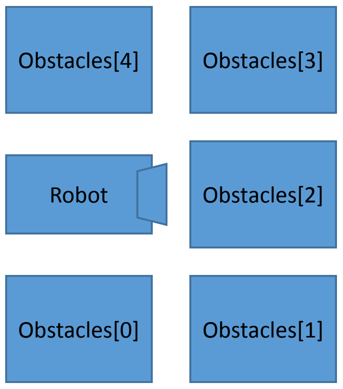
We want to control a robot through a ground station. The robot first runs a TCP server and awaits for the ground station to connect. Then, the robot periodically sends its state (x,y,heading) compared to its initial location. The robot updates its location periodically using a dead reckoning algorithm (basically it counts its wheels rotations). At the same time, it is using a rotating ultrasonic sensor in order to detect obstacles in front of it and on its sides. It sends this information to the ground station.

The ground station can issue, in direct command mode, sample commands, like “forward”, “stop”, “left”, etc. by sending strings to the robot. The ground station can also switch to automatic mode, where commands will be of higher level, like “goto x,y”. The robot, in this mode, will just go straight to x,y, if it can.

The last mode is not implemented yet! It consists in using the D* algorithm to go to a location, by considering the obstacles on the way.

The AADL design is showing that we will use the following threads:

- The main thread will create the TCP server, receive the commands, and control the mode of the robot (auto/direct). It will send the commands to the appropriate threads.
- The sending thread is collecting the information provided by the other threads (location&heading, and, when changed, current status (moving, direct command, standby), as well as, when changed, the obstacles information.
- The deadreckoning thread will be in charge of updating periodically the robot location and heading, as well as be able to reset it to a triplet (x,y,heading) when told to by the main thread.
- The scan thread will be in charge of rotating the ultrasonic sensor, reading its measurements in single shot mode, and updating the obstacles information. All this will be done periodically, at a slow period, because we need to wait for the motor rotation between the measurements.



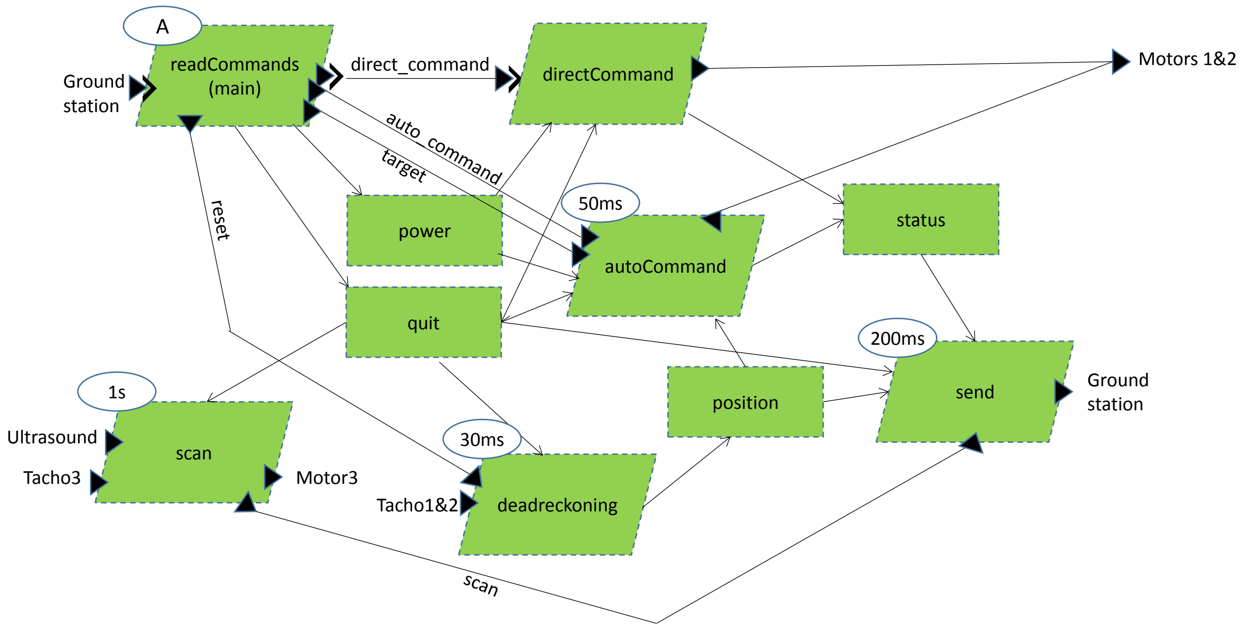
- The direct command thread has to be executed only when a direct command is received from the ground station. For example, when receiving “forward” from the ground station, it has to put the motors in RUN-FOREVER mode, using the current power value, while when receiving “right”, it has to configure the left motor to run at the current power, and the right motor to run at its opposite. When receiving stop, it stops the motors.

- The auto move thread is in charge of executing gotoxy commands. It will be ran periodically, and will execute a double proportional command (parallel command of the motors distance error to the target, mixed with differential command of the motors for angle error between current heading and required heading to go to the target).

Since we want the robot to be in a stable state when the program ends (motors stopped, ultrasonic sensor facing the front of the robot), the application must end properly, meaning that every thread should periodically consult if the application has to exit, and if so, end itself.

The main thread will thus be in charge of waiting for threads termination, before making sure that motors are stopped.

There is a little trick with the direct command task, which is waiting for messages in a mailbox: the main thread has to send it a message (it could be a “stop” message), to make sure it is waking up and terminating.



Pretty cool isn’t it?

Notice that both direct and auto commands are using the motors 1&2, therefore, readCommands has to make sure to send a stop command when switching mode, to make sure, for example, that direct command is not still activating the motors, when the autocommand is supposed to work, and vice-versa.

As usual, we will start by implementing every communication mechanism. We need 7 shared data, and 1 mailbox.

- Mailbox MB_direct_command contains a single integer (see myev3.h for proposed constants for the direct commands);
- 4 shared data can hold an integer value: MDD_power, MDD_quit, MDD_status (see myev3.h for proposed constants for status), and MDD_auto_command (we can again use the constants of myev3.h, CMD_STOP being used both in direct and auto command);

- 4 “generic” shared data: MDD_scan, MDD_position, MDD_reset, MDD_target. In C, generic means “char * buff” storage… Store anything in it, consistency is up to the programmer and not the compiler.

Shared data are to be a bit specific: several of them need to hold information telling, when reading the shared data, if the data is fresh or not. Since we can only create two types of shared data, the integer one, and the generic one, we can as well implement this mechanism for both of them. In order to help, the MDD_int type and its operations are given in mdd.h and mdd.c.

We thus need to implement 2 types of shared data (that can tell, when reading them, if the data was fresh), and 1 type of mailbox.

Some algorithms are a bit complicated for the short amount of time we have, therefore, you should look into the workers.h where you will find functions that I implemented for you (especially concerning the auto command, the dead reckoning, and the scan).

There is also a function WaitClient waiting for you in communication.h, that will wait for an incoming connection and map the socket on one input stream and one output stream, that will be used respectively by the main thread as well as by the send thread.

The documentation of the ev3dev-c library is located on http://in4lio.github.io/ev3dev-c/index.html

The file main.c is the file to complete, look for the “// TODO”. Good luck!

The ground station is a Tcl/Tk script located in the ground station folder. To run it, install ActiveTcl if necessary, then just double click on groundstation.tcl.