e-puck2 Lab Induction

ACS6501, last updated on 21.10.2019

1 First Lab Session: Setting up the PC

You will be working in groups of up to four students (the allocation is available on MOLE). When you turn on your PC, the purple bootloader screen will appear. It's important that you **select Ubuntu**, not Windows. When it boots, the password is `acselab`.

If you'd like to install the software on your own PC, see the instructions on the GCtronic website https://www.gctronic.com/doc/index.php?title=e-puck2_robot_side_development. The instructions in this document are based on this webpage. Due to limited space in the lab, each group is limited to one additional laptop.

Note: To open a terminal, use `ctrl + alt + t`
To paste into a terminal, use `ctrl + shift + v` or click the mouse wheel button

1.1 Installing JAVA 8

JAVA 8 should already be installed. Verify this with the command below.

```
update-java-alternatives -l
```

If nothing appears, move to Appendix 1 and install as indicated there.

1.2 Creating your group folder

There are more groups than PCs, so groups will need to share PCs. In order to enable this, every group should make their own folder in `Documents` with your group name, and do all your work in this folder. Choose any name you want, but *it's important there's no spaces in the name*.

At the end of the session, you should copy this folder elsewhere (eg. Google Drive, or a USB drive) and **delete it from the computer** so other groups can't copy your work. At the start of the next session, copy this folder back into `Documents`.

1.3 Installing Eclipse

The IDE used in this lab is a special version of Eclipse, preconfigured to work with the e-puck2.

1. Download the `Eclipse_e-puck2` package for Linux at this link: `https://tinyurl.com/shef-e-puck2`
2. Open a file explorer and navigate to the `Downloads` folder. Right click and press `Extract here`. When it's extracted, look inside this folder (called `Eclipse_e-puck2_Linux_11_apr_2018_64bits`) and copy the `Eclipse_e-puck2` folder to your group folder

You should end up with the following folders: `~/Documents/GroupName/Eclipse_e-puck2`

1.4 Getting source code

The code of the e-puck2 is open source and is available as a git repository. This repository contains the main microcontroller factory firmware together with the e-puck2 library. This library includes all the functions needed to interact with the robot's sensors and actuators; the factory firmware shows how to use these functions.

1. Open a new terminal and navigate to your group folder: `cd ~/Documents/GroupName`
2. Download the source code from Github to this folder: `git clone --recursive https://github.com/e-puck2/e-puck2_main-processor.git`

1.5 Familiarising yourself with the e-puck2

Take a look at Fig. 1 and familiarise yourself with the different parts of the e-puck2. In addition, take a look at the example project and try to understand the code in main.c.

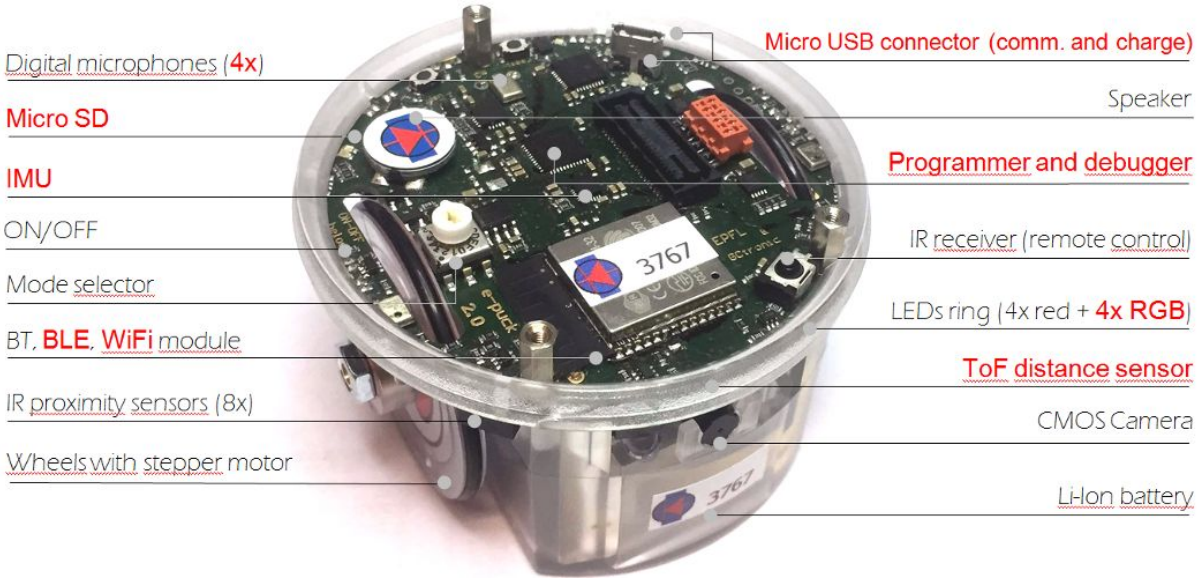


Figure 1. An e-puck2 and its components. Image reprinted from https://www.gctronic.com/doc/index.php?title=e-puck2#Finding_the_USB_serial_ports_used

Websites that you may find useful for programming the e-puck:

- EPFL (the developer of the robot) maintain an e-puck website: http://www.e-puck.org/
- In this lab, we use the e-puck library, an embedded system library for e-puck. For more information: https://github.com/e-puck2/
- GCtronic (the manufacturer of the robot) provides a mini-doc and maintain a very resourceful Wiki: https://www.gctronic.com/e-puck2.php

2 First Lab Session: Programming the e-puck2

2.1 Opening Eclipse

You can now run the `Eclipse_e-puck2` executable to launch Eclipse; it is important to open Eclipse with `sudo`, as we need root permissions to run debugger:

```
cd ~/Documents/GroupName/Eclipse_e-puck2
```

```
sudo ./Eclipse_e-puck2
```

If asked for a password, it's `acselab` again. Do not close the terminal window once Eclipse is running, or you will end up killing Eclipse itself.

2.2 Creating your own project

2.2.1 Building your own project

You can now learn how to upload your own code to the robot. For this example, you'll be using the code in the `Project_template` directory. First, add this directory to Eclipse as follows:

1. Copy the folder `Project_template` from inside `e-puck2_main-processor` to be in your `GroupName` directory and rename it how you'd like (eg. `MyProject`)
   It's important you have the following folder hierarchy:
   ```
   ~/Documents/GroupName/Eclipse_e-puck2
                   .../e-puck2_main-processor
                   .../MyProject
   ```
2. Run Eclipse (as shown in Section 2.1) and then select `File->New->Makefile Project with Existing Code`.
3. Next click on the Browse button and choose the newly copied folder `MyProject`. Choose `None` for the toolchain.
4. Click on the `Finish` button and the project is added to Eclipse.
5. Select the project root folder from the left panel (eg. `MyProject`) and go to tab `Project->Properties->C/C++ General->Preprocessor Include Paths, Macros etc->Providers` and check `CDT Cross GCC Built-in Compiler Settings`.
   Then in the textbox below ("Command to get compiler specs"), type:
   `arm-none-eabi-gcc ${FLAGS} -E -P -v -dD "${INPUTS}"`
   (Expand the window downwards if you are unable to view the text box).
6. Press `Apply and Close`.
7. Create a linked folder inside your project that links to the `e-puck2_main-processor` library. This allows Eclipse to index the declarations and implementations of the functions and variables in the code of the library. To do this:
   a. Select the project root folder and go to `File->New->Folder`.
   b. Check `Advanced >>` on the bottom.

c. Choose `Link to alternate location (Linked Folder)`.
d. Click `Browse` and select the `e-puck2_main-processor` directory (`~/Documents/GroupName/e-puck2_main-processor)`, click the `OK` button.
e. Click `Finish`.

8. Build the project by selecting one file (for example, `main.c`) of the project from the left panel and then Project->Build Project. The result of the compilation will appear in the build folder in your project folder. It should say something like "… Build Finished (took 15s.489ms)".

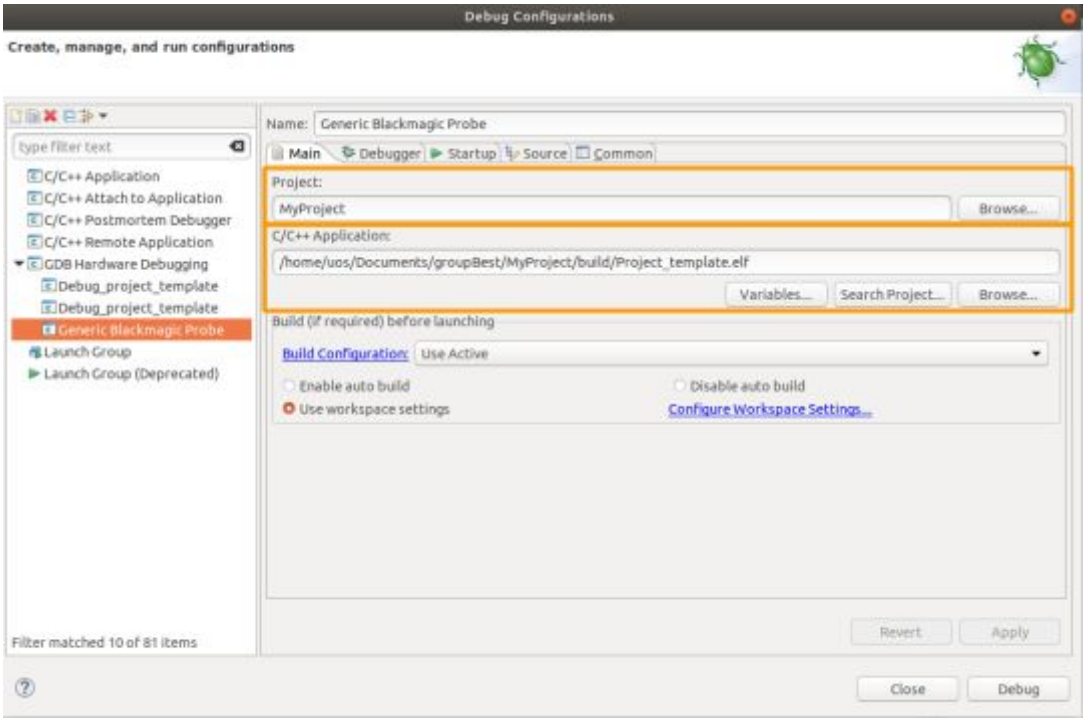
9. After you compile the project, select the project root folder (e.g. `MyProject`) and go to `Project->C/C++ Index->Rebuild` to rebuild the index (we need to have compiled at least one time in order to let Eclipse find all the paths to the files used).

2.2.2 Configuring the debugger

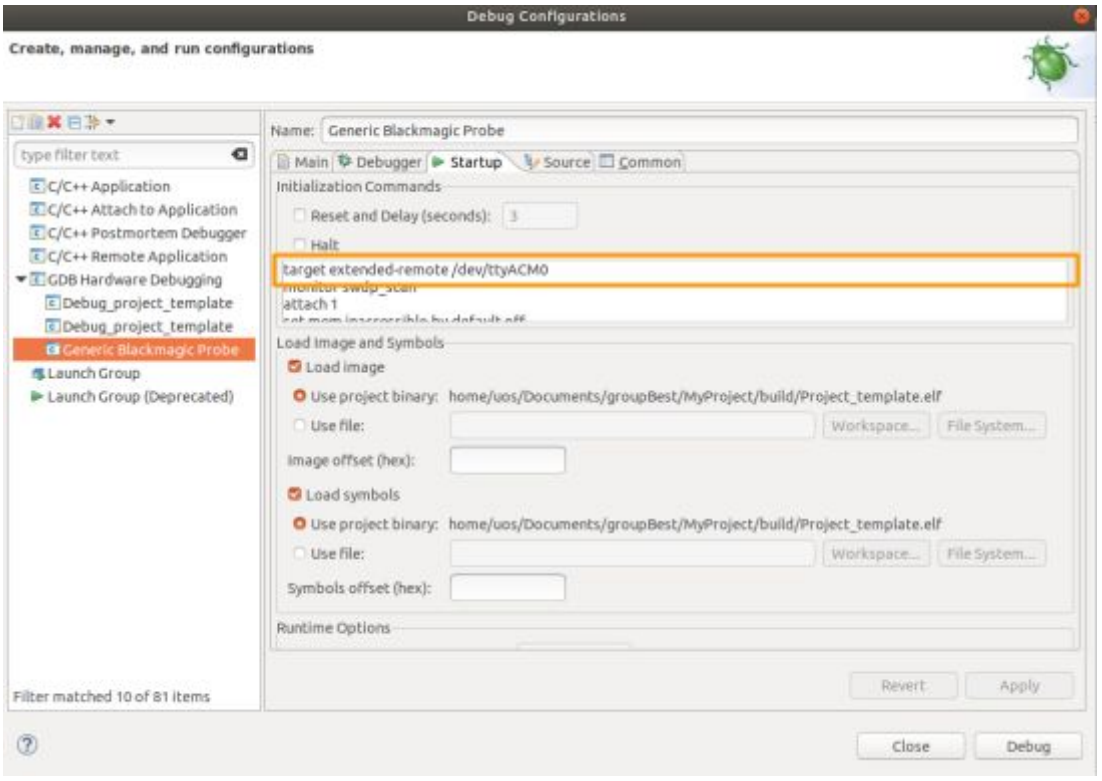
In order to upload your code, you need to use Eclipse's built-in debugger. You will need to configure settings on the debugger, you can find the debug configurations via `Run->Debug Configurations`. Once in the settings, select `Generic Blackmagic Probe` preset on the left panel. Then you need to configure two things :

1. In the `main` tab, under `Project`, click `Browse` and select your project (e.g. `MyProject`)

2. In the main tab, under `C/C++ Application`, click `Browse` and select the `.elf` file name (e.g. `~/Documents/GroupName/MyProject/build/Project_template.elf`)



3. In the `Startup` tab, you need to replace the serial port name written on the first line of the text box by the one used by the GDB Server of your robot (normally `/dev/ttyACM0`, but if this doesn't work for you when you try and download your code later on, see Appendix 2)

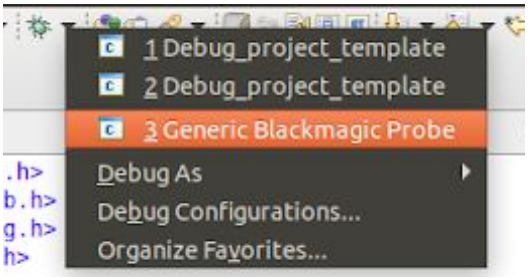


4. Press `Apply` and then `Close`.

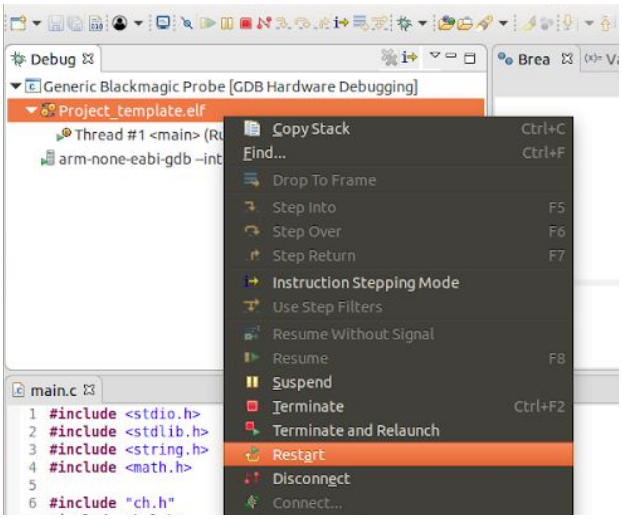
2.2.3 Uploading your project

You're now ready to upload the code of your project to the robot. To do this, follow the next steps:

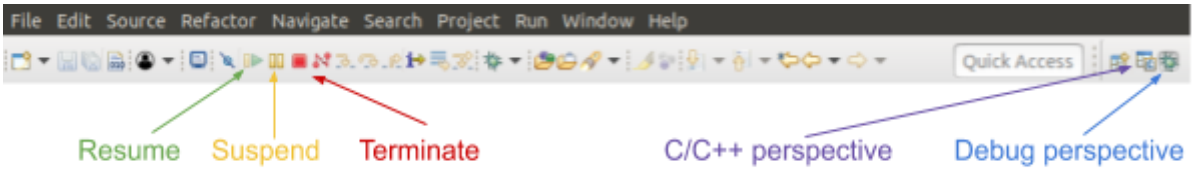
1. Connect the robot to the computer and turn it on by softly pushing the blue button below the ring. Check that the wire is inserted into the e-puck2 correctly, do not use force to push the wire down, as this might damage the e-puck2.
2. From Eclipse, launch the debug configuration previously set: from the `menu Run->Debug configurations...`, select the configuration and click on the `Debug` button.
   Alternatively you can directly select your configuration from the debugger drop-down menu as shown below.



3. When the debugging session is started, Eclipse will change the view to the `Debug perspective`. Right-click on the main process and select `Restart` to restart the program from the beginning



4. Click on the `Resume` button on top of the window to start your program. If you're just using the sample program, then it doesn't actually do anything, but the power LED will flash quickly. Now you can suspend and resume whenever you want, then when you want to modify your code again you click on the `Terminate` button and click on the `C/C++ perspective` button.



This code is now on the robot. If you disconnect it from the computer and turn the power on, it will start the code from the beginning.

Now you're ready to start making your own code!

3 Optional tasks (recommended for 1st session)

The following tasks are not compulsory, although, can help you to understand how to program the e-puck. You ideally prepare the programs at home, and use the laboratory for validation. Check the e-puck Library Cheat Sheet (available on Blackboard) for all the related functions. Please refer to the cheat sheet for a summary of the appropriate functions.

Optional Task 1: Toggle the green body LED

Modify the example program given above (e.g. `main.c` in `MyProject`) to toggle the body LEDs. You will have to use a function to wait between turning the LEDS on and off. The chThdSleepMilliseconds() function accepts a time duration in microseconds as input, 1 second is 1000 microseconds.

Optional Task 2: Control the movement with the selector switch

The e-puck has a selector on the top of it. The selector provides values from 0 to 15, some of the new e-puck models display these values in hexadecimal number system (0, ..., 9, A, ...,E). You can use `get_selector(void)` to read the current selector value.

The e-puck has stepper motors that can move up to **± 15.4 cm/s**. You can use `left_motor_set_speed(int motor_speed` function to set left wheel speed (`right_motor_set_speed(int)` for the right motor). The input `motor_speed` is an integer and bounded by **-1000** and **1000**. Thus, you should not set **15.4** to set the maximum wheel speed but **1000** instead.

Program the robot to turn on the spot, by giving both wheels the same speed but different directions, as opposite signs (e.g. 5, -5 cm/s). The body LED should blink as many times as the set selector value, after this, the robot should rotate in the opposite direction.

For instance, if the selector is set to 4, every time the body LED blinks the 4th time, the robot turns direction from clockwise to counterclockwise (or vice versa). If the selector is 0, it should never change direction.

Optional Task 3: Detect nearby objects and analyse the sensor data

Program the robot to indicate that an object is in close proximity. The e-puck robot has 8 IR receivers that can be used as proximity sensors. The proximity sensor readings depend on the ambient light. Thus, it is a good practice to use `calibrate_ir(void)` function before you get sensor readings. The range of the proximity sensors can vary in different conditions. If there is an object close to a proximity sensor, the reading is a high number.

You can send the proximity sensor readings to your computer using Bluetooth communication (UART). Please check e-puck2 Library Cheat Sheet for more information.

Whenever there is an object close to a sensor (you can define your own threshold), turn on the LED that is near that sensor. This would identify the direction of the nearby object in proxy. The range of the proximity sensors are approximately 6 cm.

For instance, assume that there is an object to the left of the robot and you set your threshold to 5 cm. If the object is close e.g. 2 cm, LED6 is switched on. However, if the object is e.g. 6 cm away, LED6 remains off. Note that only even-numbered LEDs have RGB LEDs.

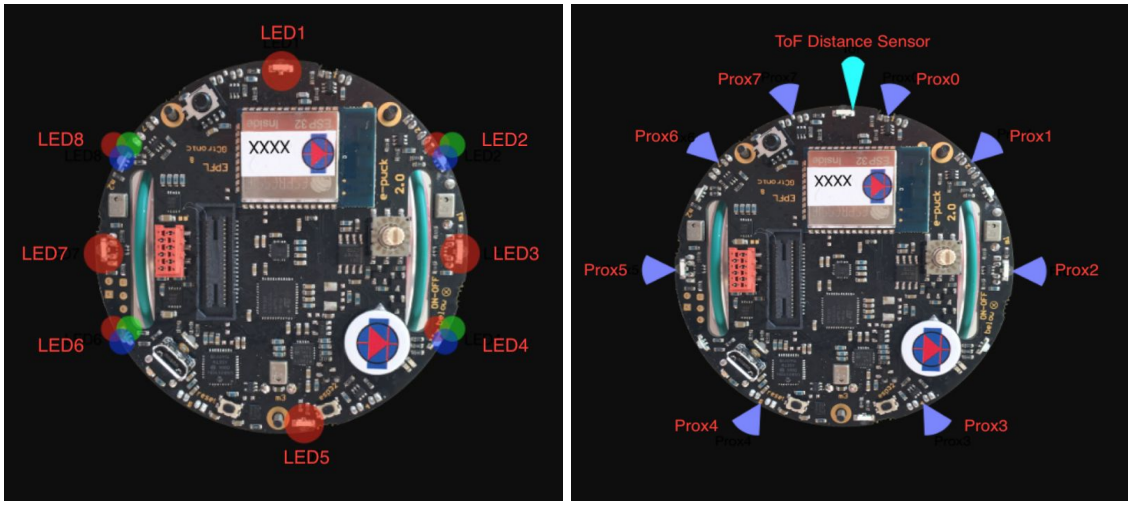


Figure 2. The e-puck2 components: ToF(Time of flight) distance sensor, LEDs, proximity sensors (image reprinted from http://projects.gctronic.com/epuck2/wiki_images/epuck2-components-position.png).

The above examples consider simple tasks for you to familiarise with the robot and programming environment. You can use the robot's accelerometer, UART communication, camera, etc. Please note that the proximity sensors needs to be calibrated or tuned each session you use a robot as the proximity readings are based on the ambient light. For more detailed explanations of the library, please visit: http://www.e-puck.org/

You will find your assessment tasks in the **e-puck Lab Assessment Briefing** document.

Good luck!

4 Saving your work and returning next session

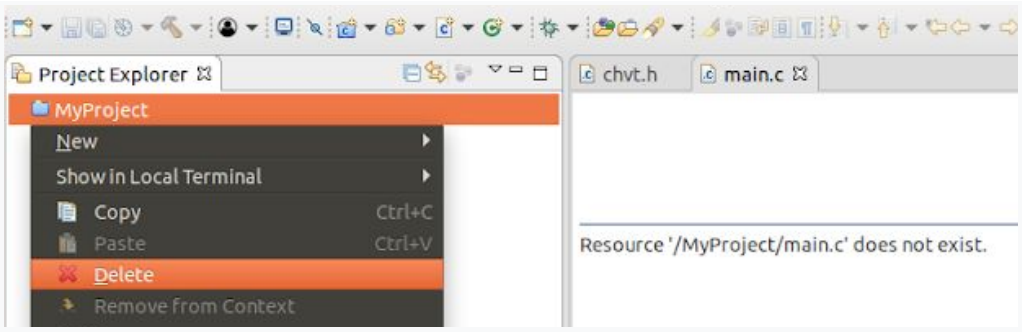
As mentioned in Section 2.1, you should do all your work in a folder which you remove from the PC at the end of the session, you should copy this folder elsewhere (eg. Google Drive, or a USB drive) and **delete it from the computer** so other groups can't copy your work. At the start of the next session, copy this folder back into `Documents`.

When you return at the start of the next session, copy this folder back to `Documents` and open Eclipse. If you were not the last person to use Eclipse on this PC, you will find that the workspace is not as you left it. If this is the case, follow the instructions below to return things to how they were.

Note: If there are any errors or issues deleting, use `sudo rm -rf ~/Documents/GroupName`

4.1 Remove existing projects

1. Remove any existing projects by right-clicking on them in the menu on the left, and selecting `Delete`



2. In the next prompt, ensure the `Delete project contents on disk` box ***is not checked***

Your Eclipse should now be back to how it started.

4.2 Replace your projects

You now need to replace your projects. To do this, follow the instructions below:

1. Select `File->New->Makefile Project with Existing Code`.

2. Next click on the Browse button and choose your project folder `MyProject`. Choose `None` for the toolchain.

3. Click on the `Finish` button and the project is added to Eclipse.

4. Select the project root folder from the left panel and go to tab `Project->Properties->C/C++ General->Preprocessor Include Paths, Macros etc->Providers` and check `CDT Cross GCC Built-in Compiler Settings`.
   Then in the textbox below ("Command to get compiler specs"), type:
   `arm-none-eabi-gcc ${FLAGS} -E -P -v -dD "${INPUTS}"`

(Expand the window downwards if you are unable to view the text box). Press `Apply and Close`.

5. Your project should remember the existence of the linked folder. If not, add this back.
6. Ensure the debugger is configured correctly for your project as in Section 2.2.2

Appendix 1 Installing JAVA 8

If for some reason you don't have JAVA installed on your PC, you can install it using the following terminal commands:

```
sudo add-apt-repository ppa:openjdk-r/ppa
sudo apt-get update
sudo apt-get install openjdk-8-jre
```

Appendix 2 Selecting the GBD server

Usually, the robot will be connected as `ttyACM0`. However, the exact numbering depends on the order in which devices were attached to the computer. To find out what's connected:

1. Open a terminal window and enter the following command: `ls /dev/ttyACM*`
2. Look for `ttyACM0` and `ttyACM1` in the generated list, which are respectively e-puck2 GDB Server and e-puck2 Serial Monitor. `ttyACM2` will be also available with the factory firmware, that is related to e-puck2 STM32F407 port.
   If you see additional ports, it's likely one of them has taken the default port number. Try a different one and see if that works.

Appendix 3 Programming in C++

If you are interested in programming in C++, there is another source template to download. Your marks will not be affected whether you program in C or C++. Follow the same steps as above, until section 2.2.1.

1. Open a new terminal and navigate to your group folder: `cd ~/Documents/GroupName`
2. In your group directory, download the C++ source template:
   ```
   git clone https://github.com/e-puck2/e-puck2_cpp.git
   ```
3. Continue to follow the steps, but note that `MyProject` is now replaced with `e-puck2_cpp`.

Note: Including headers is slightly different in C++. You should include them in `main.h`, whereas in C you include them in `main.c`. This is because there is additional information required in the header files, which you can could add to each header file you use, but has been included for you already in `main.h`.