e-puck2 Lab Induction

ACS6501, last updated on 10.10.2023

1 First Lab Session: Setting up the PC

You will be working in teams of up to two students (the allocation is available on Blackboard). 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 Appendices C (Windows) and D (Mac). This will allow you to work on the code in your own time using the preconfigured environment. Although you will not have a robot to run the code, you can check that your code compiles correctly. If you do not want to install the complete environment on your own PC, you can use a text editor of your choice to edit the necessary files. You should use Git to keep your files in sync across multiple computers (Section 4).

Notes:

- To open a terminal, use `ctrl + alt + t`
- To paste into a terminal, use `ctrl + shift + v` (or click the mouse wheel button after highlighting text)
- "Folders" are referred to as "directories" by computer scientists, so we use that name in this document.

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 A and install as indicated there.

1.2 Creating your team directory

Every team should work in their own directory in `Documents`. Choose any name you want, but it's important there's **no spaces in the name**. We call this `RepoName` here.

At the end of the session, you should back your solutions up to GitHub (see Section 4 for details on how to do this) and **delete them from the computer** so other teams can't copy your work. At the start of the next session, clone your repository back into `Documents`.

To maintain consistency, **make sure your team directory has the same name as your Git repository**. Throughout the rest of the instructions, this is referred to as `RepoName` or your team directory.

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 by opening this link in a browser and pressing `save`: https://tinyurl.com/UoS-epuck2

2. Open a file explorer and navigate to the `Downloads` directory. Right click and press `Extract here`. This may take a while, but be patient: there's a circular progress bar in the top right of the file explorer.

3. When it's extracted, look inside this directory (called `Eclipse_e-puck2_Linux64_14_aug_2020` and copy the `Eclipse_e-puck2` directory to your team directory

You should end up with the following directories:
`~/Documents/RepoName/Eclipse_e-puck2`

1.4 Getting the 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 terminal and navigate to your team directory: `cd ~/Documents/RepoName`

2. Download the source code from Github to this directory:
```
git clone --recursive https://github.com/e-puck2/e-puck2_main-processor.git
```

3. Remove the Git-related files from this directory to avoid conflicts later:
```
rm -rf 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.

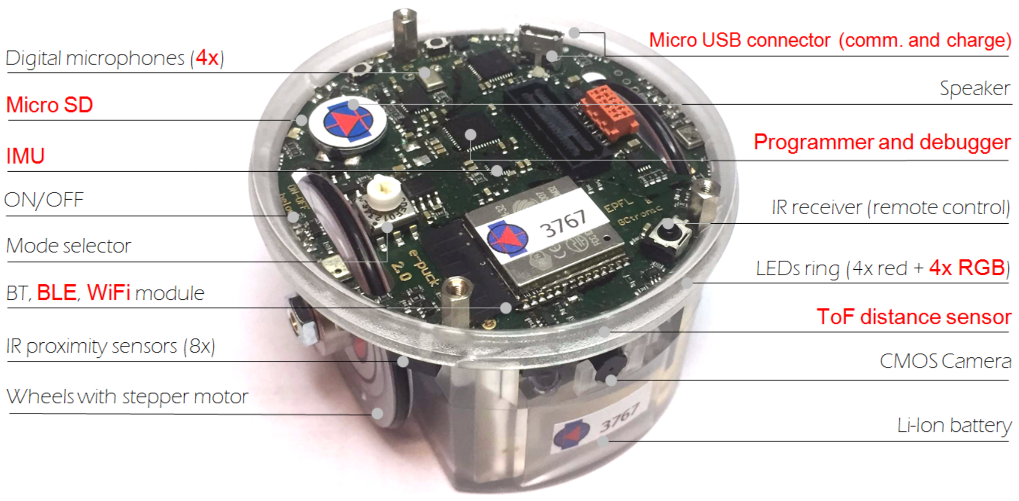


Figure 1. An e-puck2 and its components. Image reprinted from https://www.gctronic.com/doc/index.php?title=e-puck2#Finding_the_USB_serial_ports_used

Websites that you may find useful for programming the e-puck2:

- EPFL (the developer of the robot) maintain an e-puck website: http://www.e-puck.org/
- In this lab, we use the e-puck2 library, an embedded system library for e-puck2. 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

The robot is programmed with a preconfigured version of the Eclipse *Integrated Development Environment* (IDE). 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 the debugger:

```
cd ~/Documents/RepoName/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 directory `Project_template` from inside `e-puck2_main-processor` to be in your `RepoName` directory and rename it how you'd like (e.g. `MyProject`)
   It's important you have the following directory hierarchy:
   ```
   ~/Documents/RepoName/Eclipse_e-puck2
                       .../e-puck2_main-processor
                       .../MyProject
   ```
   `MyProject` should be within the `RepoName` directory, and not inside any other directory.

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, choose the newly copied directory `MyProject`, and click the `OK` button. 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 (e.g. `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/RepoName/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. This will take some time. You can monitor the progress by clicking on the progress bar icon found on the right bottom corner. The result of the compilation will appear in the build folder in your project folder. The terminal at the bottom of the screen should say something like "… Build Finished (took 15s.489ms)".

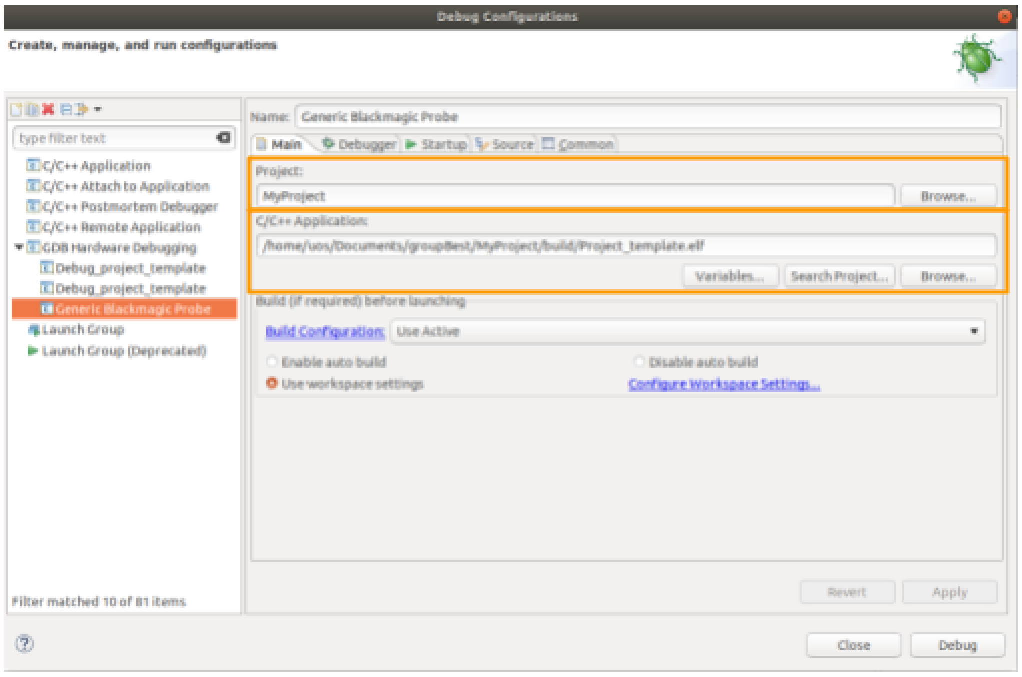
9. After you compiled the project, select the project root folder (e.g. `MyProject`) and go to `Project->C/C++ Index->Rebuild` to rebuild the index (we need to have compiled at least one time in order to let Eclipse find all the paths to the files used).

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 :

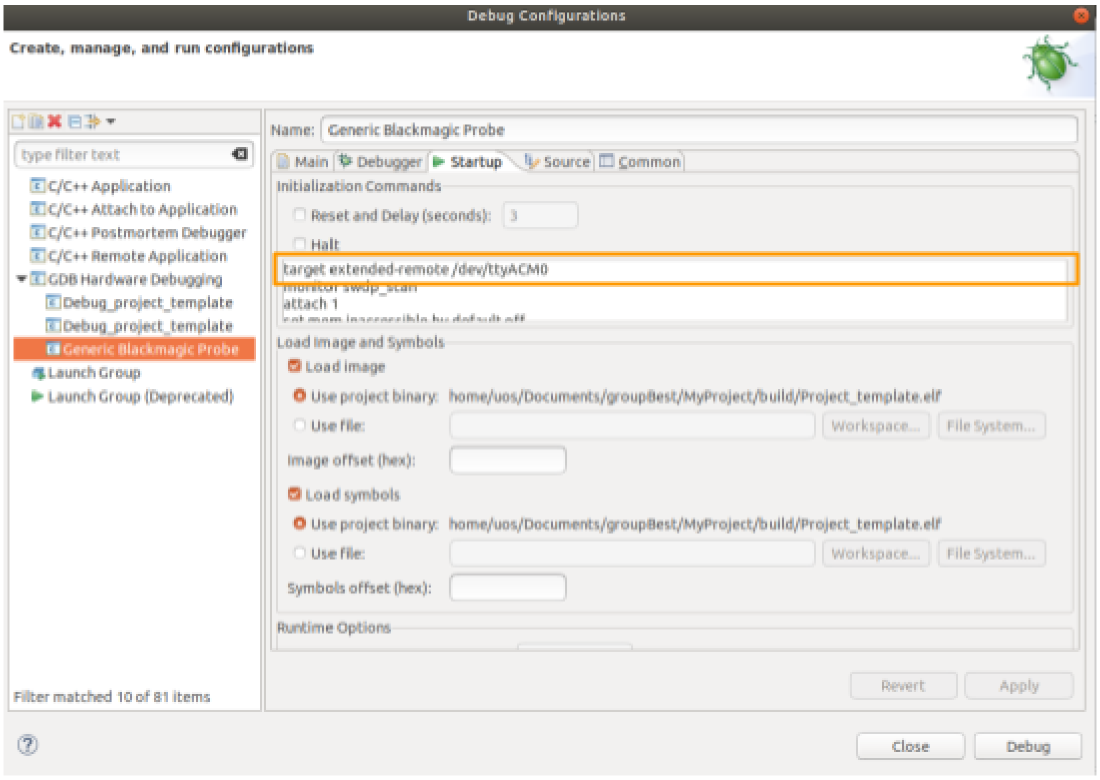
1. In the `main` tab, under `Project`, click `Browse` and select your project (e.g. `MyProject`)

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


Name: Generic Blackmagic Probe
Project: MyProject
C/C++ Application: /home/uos/Documents/groupBest/MyProject/build/Project_template.elf
Build (if required) before launching
Build Configuration: Use Active
Enable auto build
Disable auto build
Use workspace settings
Filter matched 10 of 81 items

3. In the `Startup` tab, 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 B)



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
  - Debug_project_template
  - Debug_project_template
  - Generic Blackmagic Probe
- Launch Group
- Launch Group (Deprecated)

Filter matched 10 of 81 items

Name: Generic Blackmagic Probe

Main | Debugger | Startup | Source | Common

Initialization Commands

Reset and Delay (seconds): 3

Halt

```
target extended-remote /dev/ttyACM0
monitor swdp_scan
attach 1
set mem inaccessible-by-default off
```

Load Image and Symbols

[✓ Load image]

[✓ Use project binary:] home/uos/Documents/groupBest/MyProject/build/Project_template.elf

Use file: Workspace... File System...

Image offset (hex):

[✓ Load symbols]

[✓ Use project binary:] home/uos/Documents/groupBest/MyProject/build/Project_template.elf

Use file: Workspace... File System...

Symbols offset (hex):

Runtime Options

Revert Apply

Close Debug

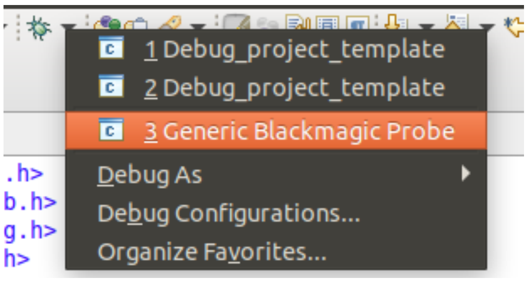
4. Press `Apply` and then `Close`.

2.2.3 Uploading your project

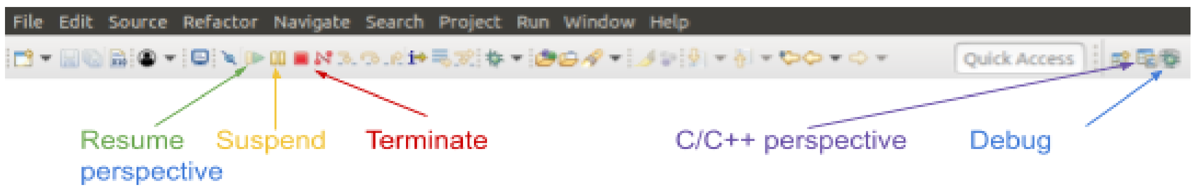
You're now ready to upload the code of your project to the robot. To do this, follow the next steps:

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 robot.

2. From Eclipse, launch the debug configuration previously set from the debugger drop-down menu as shown below; this appears by pressing the arrow next to the bug icon.



3. When the debugging session is started, Eclipse will change the view to the `Debug perspective`. 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 to indicate the robot is talking to the debugger. 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. **It is very important that you don't just unplug the robot from the PC while it is still debugging**. Doing so could cause the firmware on the robot to be corrupted, which is hard to recover from. **Instead, when the green LED on top of the robot starts to flash you should press the red `terminate` button in the debugger**. This will restart your code independent of the computer (indicated by the green LED turning steady on), so you can now unplug it as you like.

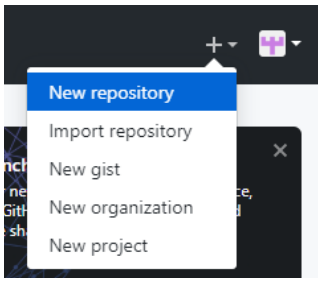
Before you get started with developing your own code, it's important to set up Github as this can take some time so it's best to get it done well before the session. See the next section for instructions on how to do this.

3 Saving your work

As mentioned in Section 2.1, you should do all your work in a directory which you remove from the PC at the end of the session. You will back this up to GitHub,or a different Git hosting website if you prefer. This enables you to continue to work from home, remove your work from the PCs so other teams don't copy you, and also introduces you to Git, which is used widely in industry and academia. For more information on what Git is and why it is useful, see this link. *Before proceeding, ensure you followed all the steps in section 1.4, including removing the files created by Git when you cloned the existing e-puck2 source code repository.*

3.1 Creating a new git repository

First you must create a repository on GitHub. To do this, log into your account and press the `New repository` button in the top right, as shown below.



Give it the **same name as the directory you've been working in**, referred to as `RepoName` in this documentation, and a description so you remember what it is. Make sure you set it to be a private repository before pressing `create repository`. Don't bother to initialise it with a `README`, `.gitignore`, or licence. At the top of the next page is a blue box showing your repository URL. Copy this for later, ensuring `HTTPS` is selected, not `SSH`.

We can now prepare your directory to be pushed to GitHub. To do this, open a terminal and follow the instructions below:

1. Navigate to your `RepoName` directory: `cd ~/Documents/RepoName`

2. Initialise this directory as a Git repository: `git init`

3. Tell Git the location of the repository you created using the URL you copied earlier:
   ```
   git remote add origin https://YOUR-REPOSITORY.git
   ```

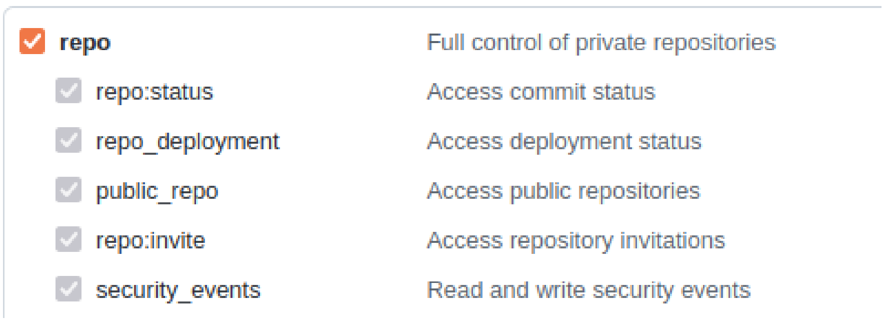
4. Now you need to tell Git who you are. Since we share these computers, you should not do this globally, but instead ensure it's only valid for this repository using:
   ```
   git config user.name "Your Name"
   git config user.email "Your email"
   ```

3.2 Generating a new access token

Your repository is private, so you will need to authenticate yourself before you can push any code to it. A recent update to Github means that you cannot simply use your password, but instead must generate a *personal access token*. These are like temporary passwords that add an additional layer of security to your work. In order to do this, follow these steps:

1. Log into Github, and in the top-right corner click your profile photo, then select `Settings`

2. In the left hand sidebar, go to `Developer Settings` then `Personal Access Tokens`
3. Press `Generate new token`, give it a useful name like “Lab session 1”, and set it to expire after 7 days. You could make it last longer, but you will likely forget it by the end of the session!
4. The next section asks you what operations you would like the token to be able to perform. In our case, we only need to select `repo`, as shown below.



✓ **repo**	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
✓ security_events	Read and write security events

5. Press `Generate token`. Your token will appear on the next page, but will not be visible once you leave this page. Copy it straight away, and paste it somewhere memorable, such as a new text document.

You will need to repeat this procedure whenever you push changes to or pull data from git, unless you can remember the previous access token.

3.3 Pushing to git the first time

Now you are ready to push your initial work to GitHub. We do this by first telling Git what changes you’ve made, then how to label these changes, and finally pushing them to GitHub. This is done with three commands:

```
git add .
git commit -m "Initial commit"
git branch -M main
git push -u origin main
```

When prompted for a password, use the personal access token you generated earlier. If you now refresh your GitHub webpage you should now see your files in the repository. Have a quick check to see that everything’s there.

At the end of the session, please **delete your work from the computer** so other teams can’t copy your work. If there are any errors or issues deleting, use `sudo rm -rf ~/Documents/RepoName`

3.4 Pushing additional changes to or retrieving changes from GitHub

In later sessions you will push more work to GitHub as you develop your code. Git is a good way to track what changes are made as it remembers the history of commits made to the repository, easily tracking changes and allowing you to also work collaboratively in your own time without conflicting with each other.

To push additional changes to the repository, you will first need to have an access token as described in Section 4.2. Once you have this, navigate back to your `RepoName` directory and tell Git who you are:

```
git config user.name "Your Name"
git config user.email "Your email"
```

Then you can run the following commands as when we pushed our changes the first time:

```
git add .
git commit -m "Commit notes"
git pull
git push
```

You can change the `Commit notes` to be whatever you'd like, such as `Work from session 2` or `Fixed LED bug`. Human-readable notes like this are very valuable to tracking your work. Pulling and then pushing your changes multiple times throughout the session is a good way to maintain version control, which could be invaluable should you need to revert to a previous version.

If you wish to get the latest version of what is on GitHub (which the team member you are working with may have changed), then run:

```
git pull
```

Now you're ready to start writing your own code! See the accompanying Cheat Sheet document for information about what functions you can use to make your robot move, read sensor data, etc. Before tackling the assessed tasks, have a go at the unassessed tasks in the next section to familiarise yourself with the robot and its capabilities.

4 Unassessed tasks for the first session

The following tasks are not assessed, but they will help you to understand how to program the robot. You may prepare the programs at home and use the laboratory for validation, though prior to testing a complex program, it is always good to test something basic, to make sure the robot is actually working. Check the e-puck2 Library Cheat Sheet (available on Blackboard) for all the related functions. Please refer to the Cheat Sheet for a summary of the appropriate functions.

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, where 1 second is 1000 microseconds.

Task 2: Control the movement with the selector switch

The e-puck2 has a selector on the top of it. The selector provides values from 0 to 15 in the hexadecimal number system (0, ..., 9, A, ...,E). You can use `get_selector(void)` to read the current selector value.

The e-puck2 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 indicated by the 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.

Task 3: Detect nearby objects and analyse the sensor data

The e-puck2 robot has 8 IR receivers that can be used as proximity sensors (Figure 2). The proximity sensor readings depend on the ambient light, so it's good practice to use the `calibrate_ir()` function before you get sensor readings. If there is an object close to a proximity sensor, the reading is a high number. To get an idea of what values are produced, write some code to send the proximity sensor readings to your computer. The robot can communicate with your computer via Bluetooth, or the USB cable. Both ways of communication are explained in the Cheat Sheet, but we recommend Bluetooth. Try to print something to the console that looks like that shown below, which updates every 50 ms:

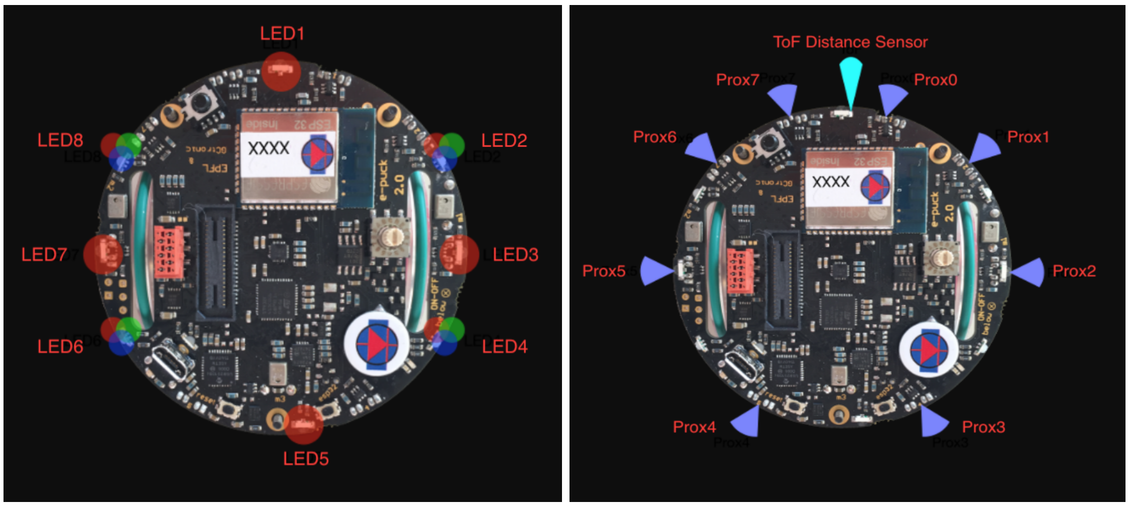


Figure 2. The e-puck2 components: ToF (Time of flight) distance sensor, LEDs, proximity sensors (image reprinted from http://projects.gctronic.com/epuck2/wiki_images/epuck2-components-position.png).

```
-----------------------
Sensor 0: 106
Sensor 1: 151
Sensor 2: 2147
Sensor 3: 1
Sensor 4: 1
Sensor 5: 59
Sensor 6: 3658
Sensor 7: 330
-----------------------
```

The above examples consider simple tasks for you to familiarise with the robot and programming environment. You can use the robot's accelerometer, camera, etc. Please note that the proximity sensors need 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-puck2 Lab Assessment Briefing** document.

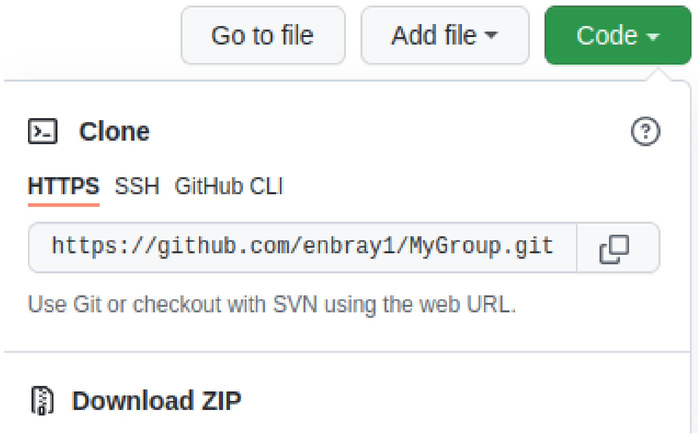
Good luck!

5 Returning next session

When you return the next session, you will need to pull your work from GitHub again.

5.1 Cloning from GitHub

When you return at the start of the next session, you will need to clone your work from GitHub. Open up GitHub and go to your repository. **Press the `Code` button shown below and copy the URL for later**.



```
https://github.com/enbray1/MyGroup.git
```

Use Git or checkout with SVN using the web URL.

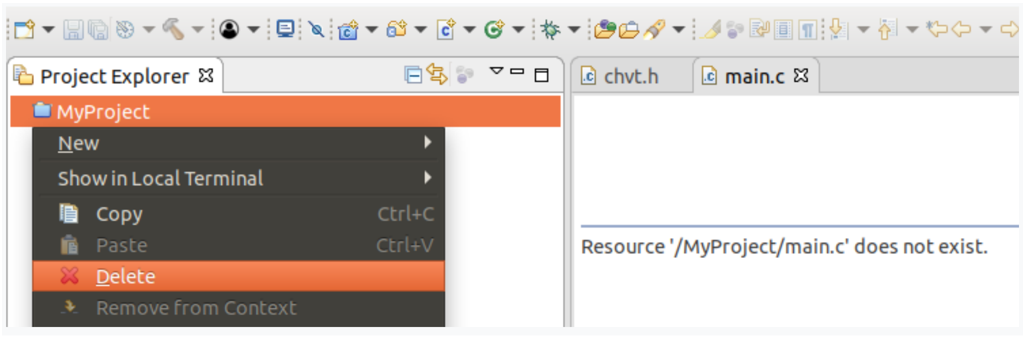
You will also need a personal access token (see Section 3.2). Now open a terminal and go to `Documents` with `cd ~/Documents`. You can now download your code again by simply using `git clone https://YOUR-REPOSITORY.git` and pasting the URL you just copied. All your files should now be restored back where you left them, in a directory called `RepoName` exactly as before. If your repository name and the directory you were working in during the first session have different names, then you may have issues with certain linked files. Talk to a GTA about this.

5.2 Restoring 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.

5.2.1 Remove existing projects

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



Resource '/MyProject/main.c' does not exist.

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.

5.2.2 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 A 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 B 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 C Installing on Windows

Installing on Windows is relatively straightforward. First, you need to install the correct version of JAVA.

1. Go to OpenJDK download page: https://adoptopenjdk.net/releases.html (if prompted, close pop up window indicating that AdoptOpenJDK has moved)
2. Choose the `OpenJDF 8 (LTS)` version and `HotSpot` JVM. Select `Windows` as the operating system and `x86` as the architecture. Download the `.msi` file from the list.
3. Double click on the downloaded file to install it.

You now need to install the Eclipse IDE as before. We use a different version specifically for Windows, available here: https://tinyurl.com/sheff-epuck2-windows

1. Once the download is complete, extract the .zip file. It doesn't matter where you extract it to (as long as it's not in `Program Files (x86)`), but make sure there aren't any spaces in the path. For example, `C://uni_stuff/Eclipse_e-puck2` is fine, but `C://uni stuff/Eclipse_e-puck2` isn't
2. You can open Eclipse by double-clicking on `Eclipse_e-puck2.exe`, or make a shortcut to this file and put it in a more convenient location.

In order to use Git, you'll need to download it from https://gitforwindows.org/. You can then run the `git bash` program from the Windows Start menu. This will bring up a terminal window where you can use the same commands we used before to navigate directories, clone repositories and so on. It's recommended to keep your Git repository directory separate from the `Eclipse_e-puck2` directory to avoid conflicts between the Windows and Linux versions.

You should now be able to create or load projects as described in Sections 2 and 4. Don't worry about configuring the debugger on Windows, as you will be able to use the Linux PC in the lab to program your robot. If you would like to use your own laptop in the lab to program the robot, *lab staff won't be able to offer support* as the success of this can be very dependent on your computer's specific configuration.

Appendix D Installing on Mac

To compile your code on Mac, you need to have the `Command Line Tools` installed. Open the Terminal app and run the command `xcode-select --install`. This will install the necessary tools that allow you to build code on your Mac. Otherwise, it will tell you that it is already installed.

Next you need to install the correct version of JAVA.

1. Go to OpenJDK download page: https://adoptopenjdk.net/releases.html (if prompted, close pop up windowindicating that AdoptOpenJDK has moved)
2. Choose the `OpenJDK 8 (LTS)` version and `HotSpot` JVM. Scroll down to `MacOS` for the operating system. Download the `.pkg` file that says `JDK` from the list.
3. Double click on the downloaded file to install it.

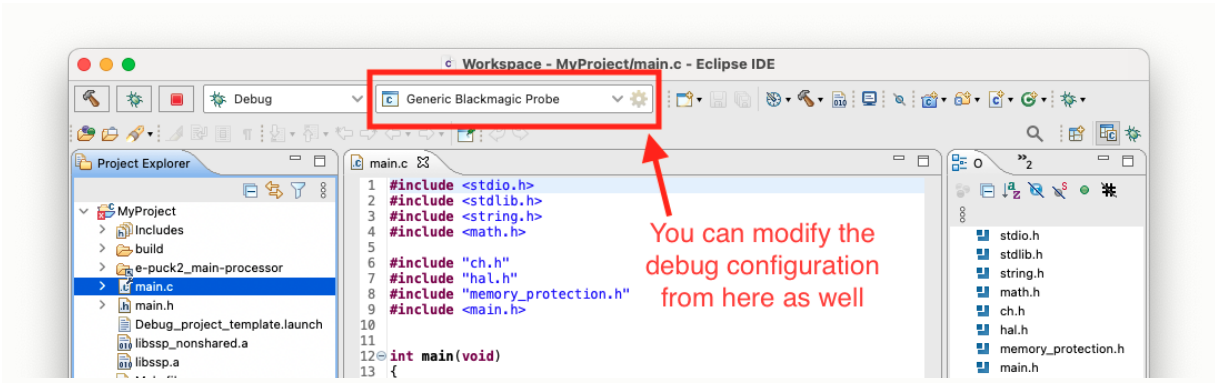
You now need to install the Eclipse IDE as before. We use a different version specifically for MacOS, available here: tinyurl.com/shef-e-puck2-mac

1. Click on the .dmg file and install Eclipse_e-puck2 for Mac. A window will pop up that tells you to drag Eclipse to your Applications folder.
2. After you complete Section 2.2.1, you should have a folder named something like `RepoName`. Navigate to this folder by running `cd ~/Documents/RepoName/` in the Terminal.
3. Open Eclipse from the Terminal with root permission using the following command in Terminal.
   `sudo /Applications/Eclipse_e-puck2.app/Contents/MacOS/Eclipse_e-puck2`
   Enter your password to open Eclipse with root permission. You may see a warning message regarding malware. If so, press cancel. Then open your MacOS System Preferences, go to Security & Privacy, click on the lock to make changes, and under "General", allow the Eclipse_e-puck2.app to be launched, by clicking "Open Anyway".

Git should already be installed as it is part of the `Command Line Tools`. You can check if it is installed by running the command `git --version` in the Terminal. If you don't have it installed already, it will prompt you to install it.

Now follow the steps in **section 2.2.1**. In **section 2.2.2**, there is a slight difference in configuring the debugger on Mac, as explained below.

In the debug configuration setting in `Run->Debug Configurations`, the `Generic Blackmagic Probe` preset may not be visible. You can still configure this preset by selecting it from the drop-down menu shown in the red box below (click the gear icon).



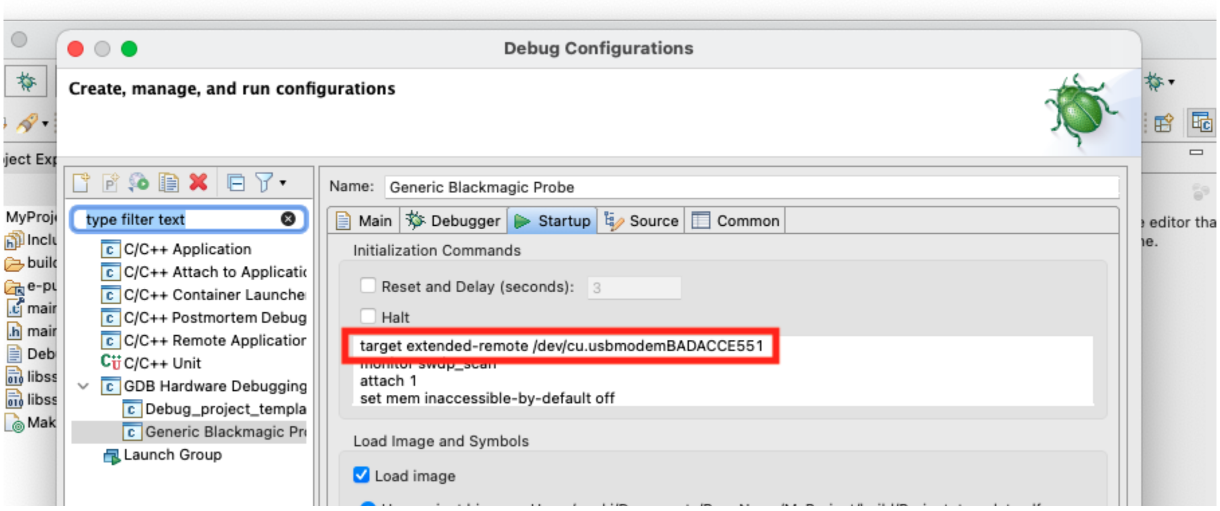
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "ch.h"
#include "hal.h"
#include "memory_protection.h"
#include <main.h>


int main(void)
{
```

In section 2.2.2 step 3, you need to replace the serial port name by the one used by the GDB Server of your robot.

1. Power on the e-puck2 and connect it to your Mac.
2. Open Terminal and run `ls /dev/cu.usbmodem*`
   Look for two `cu.usbmodemXXXX`, where `XXXX` will be different for your Mac. They should end with a number at the end.
3. Among the two names that you find, note the serial port name ending with a lower number and add it to the configuration in the format `/dev/cu.usbmodemXXXX` as shown below.
4. Select `Apply` to confirm the change.



Name: Generic Blackmagic Probe

Initialization Commands

Reset and Delay (seconds): 3

Halt

```
target extended-remote /dev/cu.usbmodemBADACCE551
monitor swdp_scan
attach 1
set mem inaccessible-by-default off
```

Load Image and Symbols

Load image ✓

Appendix E 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 terminal and navigate to your team directory: `cd ~/Documents/RepoName`

2. In your team 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 could add to each header file you use, but has been included for you already in `main.h`.

Appendix F Debug Using Breakpoints

You can add breakpoints to certain lines in your code to make the program pause. This can be useful when you want to know the value of a variable at that point.

To use breakpoints in Eclipse:

1. Add or remove a breakpoint by double clicking the area left of the line number.
2. Run the debugger as described in Section 2.2.3.
3. The execution of the code will stop at the breakpoint.
    a. Use the Variables tab to see current variables and their values that are within scope.
    b. Use the Expressions tab to view the value of a particular expression e.g. proc[0]
    c. If the value says `<optimized out>`, it means the compiler has removed values that appear in several places. The values can still be viewed by stepping into the functions with the correct scope. For example, to view the proximity sensor readings, press the Step Into button (it's near the Terminate button in the Debug perspective) when the program reaches `get_prox()` to enter the function.

You can control the execution by pressing on the resume, step into, step out buttons found in the toolbar.