User guide

Thymio-II architecture

Billod Nicolas
Lorthioir Guillaume

Jan-May 2017

Summary

1 Introduction

This document is a user guide for the code we wrote for a project during our first year of Master at the Pierre et Marie Curie university, France. The name of this project was "Control architecture for a fleet of mobile robots" and our supervisor was Aurélie Beynier from the multiagent group (SMA) of the LIP6 (Laboratoire d'Informatique de Paris 6).

First, we will explain how to install the necessary packages for running our code, and then we will talk about its architecture and describe the modules that it implements

2 Getting ready to use

The first thing to do before even trying to compile is to get the GDBus library brought by the GLib package. But, most likely, it has already been installed with your Linux distribution. Then, you should be able to compile the code using `make` but not execute it yet.

To do so, you will have to go to `https://www.thymio.org/en:linuxinstall` and follow the steps to install the Aseba package for your distribution. We will walk you through for the Raspbian one (jessie-raspbian distribution on our Raspberry Pi 3).

Raspberry Pi

Download `libdashel_1.1.0_armhf.deb`, `libenki_2.0_armhf.deb`, and `aseba_1.4.0_armhf.deb` packages from the thymio website. Then, use this command line to install them:

```
$ sudo dpkg -i packagename.deb
```

starting with `libdashel` and `libenki`, and then `aseba`. You may encounter some errors if your distribution is not up to date, to fix it you can try to execute the command lines:

```
$ sudo apt-get update
$ sudo apt-get -f install
```

and then try again.

Now that the packages are installed, you need to be connected in ssh with the -X for that part (or directly to the raspberry pi):

```
$ asebastudio "ser:device=/dev/ttyACM0"
```

It should launch the Aseba Studio with the Thymio II, assuming he is under `/dev/ttyACM0` (the `lsusb` command can be useful). Though, to run our code it is not the Studio we are interested in but the Asebamedulla program. Now, you have two possibilities: either launch a new terminal and execute

```
$ asebamedulla "ser:name=Thymio-II"
```

or just do it with a `&` but then you will have to kill its running processor. Once it is running, you can execute our code normally.

3 Code architecture

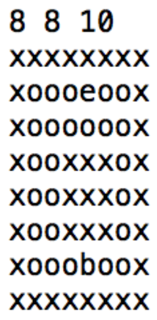
We organized the code using an architecture based on levels. In each level, there is at least one module (group of similar functions) which uses modules from sublevels; expect for Level 0 where modules do not need anything else. The `include` directory contains all the headers.

The `main.c` file contains the main function in which calls to modules are made. Most of them are already there in comments. To compile everything, use the command line `make`, and execute the generated `test` file.

3.1 Level 0

This level is used to collect data from the sensors, and for sending directives to the robot. Basically, getters and setters for motors, sensors and buttons, but also some functions for handling the maps.

A **map** file needs to have a precise form. It is a text file with on the first line the number of cells in width, in height, and the size of a cell in centimeters. Then, the mapping should start on the second line. An `x` means that there is an obstacle on the cell and an `o` means it should be free. The `b` is for the initial position of the robot and the `e` is for the objective (end point). Figure 1 shows an example of a map file. At the moment, those files are located with the `main.c` and are only needed by the `sizeMap()` and `toMap` functions from Level 0 (to get the width, the height, the size of cells and to turn it to a table of characters).



```
8 8 10
xxxxxxxx
xoooeoox
xooooooх
xooxxxox
xooxxxox
xooxxxox
xooobоox
xxxxxxxx
```

Figure 1: Content of a map file

The following files are located in this level.

- `proxSensor.c`: Getters for horizontal, ground ambiant, ground reflected, and ground delta sensors.

- `otherSensors.c`: Getters for temperature, accelorometer, and sound intensity values.
- `buttons.c`: Getters for forward, backward, left, right, and center button states (pushed or not).
- `motors.c`: Getters and setters for motors values. It goes from -500 to 500. A value of 500 approximately corresponds to a linear speed of 20 cm/s.[1]
- `map.c`: A getter from a map file for width, height, size values, and a table of characters (maximum of TAILLE_Max characters). Also a function to write a map file from those values.

3.2 Level 1

This level is used to detect obstacles, and also to compute a path from the `b` point to the `e` point. The first module calls a lot of functions from level 0, whereas the second one doesn't; it is only there because it requires a map (obtained using the map module from level 0) as an entry.

- `detection.c`: Module to dectect if there is an obstacle in front of the robot, on its back, its left, or its right. Also a function for the ground sensor and one to dectect sound.
- `computePath.c`: Compute a path to the objective from a map using the A* algorithm. It returns a table with the indexes of cells in the map to follow to reach the target.

3.3 Level 2

This level is used for all the robot's movements. The linear trajectory movement has been implemented three times, once with no paramaters except for the speed, once with a distance (centimeters), and once with distance and check for obstacles.

- `movement.c`: stop, linear movement with different paramaters (e.g. distance to travel, with a check for obstacles), and turn left or right.

3.4 Level 3

This level is used to orientate the robot. To be able to know which way the Thymio was facing in the world, we used the orientation circle shown in Figure 2. So, if its orientation is 0 it is facing the top side of the map file (cf. level 0), if it is 90 it is facing the left side, and so on.

We also used this level to implement a randomwalk module allowing the robot to wander around.

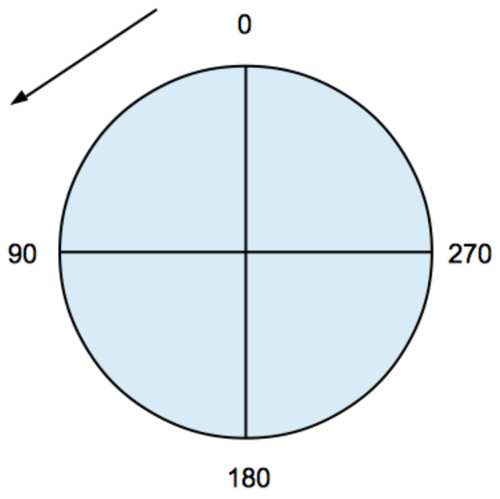


Figure 2: Orientation used by the robot

- `advanced.c`: A randomwalk with its own obstacle avoidance.
- `orientation.c`: Orientate the robot in the right direction (orientation) using Level 2 turn right and turn left functions.

3.5 Level 4

This level is used for executing a path computed in Level 1. It also needs the orientation and the movement modules from Levels 3 and 2.

- `pathExecution.c`: Either execute a path without checking for obstacles or with checks (doesn't update the map file yet but only the robot's internal map).

3.6 Level 5

This level is used for the exploration. It is the highest one so far and it needs the map module (Level 0), the detection module (Level 1), the path computing module (Level 1), the orientation module (Level 3) and the path execution one (Level 4) to work.

The robot will create a map file with a square grid with a given width. It will assume he is in the center cell and will start exploring from there, until there is no unexplored cell or the remaining ones are not accessible.

- `exploration.c`: Execution of an exploration ending with the generation of a map file.

4 Contact

If you have any questions about this user guide or the code itself, don't hesitate to contact us at gui.lorthioir@gmail.com and nicolasbillod@gmail.com. The written report can also be of help because it includes some detailed algorithms we used.