Université Catholique de Louvain

École Polytechnique de Louvain

LINGI2146 - Mobile and Embedded Computing

- May 16, 2016-

Instructor : Sadre Ramin

Game controller - Project report

Group 10 :

Deconinck Guillaume
Mokaddem Sami
Vaessen Tanguy

1 Introduction

Have you ever dreamed of having your own, customizable game controller ? We have and that's what we have been working on during this project.

This report will first introduce the project to the reader: how did we build it ? What is the phidget that we used ? How does it work ?

Then we will explain how we implemented the solution on the Z1 and the Re-mote. But also the implementation of our nice graphical interface written in Python.

Before jumping to the conclusion, we will explain a bit the difficulties we met and show some measurements of the speed reaction in wired and wireless modes.

And finally, we will end this report with a conclusion of what we have learned and some possible improvements we could have made.

2 Project description

The goal was to use the Z1 as a game controller in two modes: wired and wireless. To do so, we attached a Joystick phidget[1] to the Z1 and used the accelerometer integrated in the Z1. For the accelerometer, we only used the X and Y axis (since you don't drop your game controller very often, unless you're sore loser). This combination gave us 8 "directions":

- The fourth of the joystick: LEFT, RIGHT, TOP, BOTTOM
- The fourth of the accelerometer: WEST, EAST, NORTH, SOUTH

If the Z1 is plugged directly to the computer, it is in **wired** mode and send directly the directions to the USB interface. The wired mode is advertised by the BLUE led. Otherwise, it is in **wireless** mode, advertised by the GREEN led. In this case, the Re-mote must be connected to the computer and it acts as a relay between the computer and the Z1.

Another feature is the battery level that is checked every five minutes. If the battery level gets low, the RED led is turned on.

Finally, we build a GUI to bind any keyboard input to any of the 8 directions, show the battery level and the current mode (wired or wireless).

3 Our solution

In this section, we explain the technical details of our solution. We have three parts to present: the implementation of the Z1, acting as the game controller, that is the main part of our solution. The Re-mote that acts just as a relay and finally the GUI interface that receives and interprets the messages received from the Z1.

3.1 Z1 implementation

We built the Z1 solution incrementally, mostly using the well-known "Pair programming" technique. The first thing that we developed was to receive and analyze the input of the Joystick phidget. This was pretty

[1] `http://www.phidgets.com/products.php?product_id=1113`

straightforward since the two axis of the Joystick are simply represented by a voltage arriving on a port of the Z1. After taking note of the default values, we were able to interpret the variation of voltage and if they were higher than a threshold (set to 200), we consider that the joystick is pointing to a direction.

Our next move was to send information to the computer via the USB connection. We first wanted to use Tunslip to communicate with the computer but we weren't able, on the computer side, to connect a program to the **tun0** interface and retrieve the packets, even if the Z1 was able to send packets to the tunnel. Hence, we moved to another solution: simply use the USB connection as a raw serial interface on which the Z1 (or the Re-mote) can throw some lines of text that can be retrieved by a python program on the computer side. We found a snippet of code on Internet[2] that allow us to easily connect a python script to the serial interface and parse its content.

Once we were able to receive the joystick directions on the computer, we start looking at the wireless functionality. We thought about using 6LoWPAN but finally used Rime since it consumes less power. We created a **send** function that either print a message (to the USB interface) or broadcast it on the channel 129 of Rime, depending on the mode (wired or wireless).

To change the mode, we created a new process called **test_button_process** that wait for the user to be pushed and switch the mode (and the LED). The BLUE led indicates the wired mode and the GREEN one, the wireless mode.

We also added another process: **test_battery_process** to update the battery status and light on the RED led if the battery level gets low. We had some trouble implementing this feature, as it is explained in the Difficulties section.

We wanted to use the accelerometer to add new directions to our game controller. After looking at an example, it was really straightforward to implement: when idle, the Z1 is at a value of 0 for the **X axis** and **Y axis**. Moving the Z1 in one of the direction increase positively or decrease negatively those value. Thus, we just retrieve the value of the accelerometer in the X axis and Y axis, and if those value are higher than a defined threshold, we send the direction.

Finally, we optimized the communication with the Re-mote and the computer by putting all the current directions into one single string (example: "blne" indicates that the joystick is on the bottom and the left and that the accelerometer is on the north and the east) instead of sending one individual packet for each direction.

3.2 Re-mote implementation

The implementation of the Re-mote was quite easy. It simply needs to listen to the broadcast messages sent by the Z1 and transfer that data to the serial port. More precisely, the Re-mote listens to the channel 129 in Rime, and for each message, print the data (which is automatically send to the serial port). This part is inspired by the broadcast example in the Rime folder.

3.3 Python GUI implementation

3.3.1 Features

In order to fully enjoy our controller, we designed a Key binder for the Z1 mote. A screenshot of the GUI is shown on the picture 1. Obviously, as you can see, it permits to bind the 2-axis directions with any keyboard

[2] `http://anrg.usc.edu/contiki/index.php/Interfacing_with_Python`

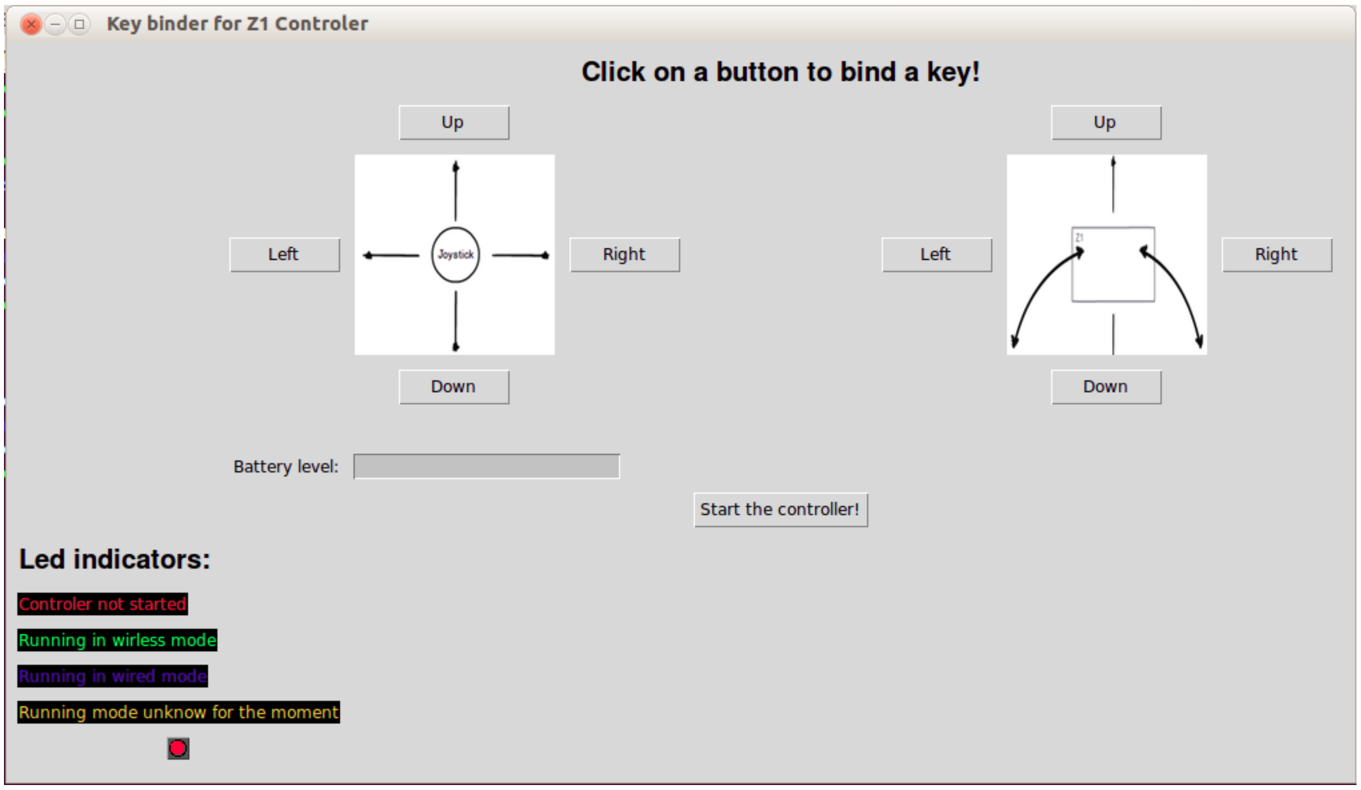

Figure 1: Our graphical user interface

key. Identically, the accelerometer have the same feature.

In addition, The software provides two indicators:

- The battery level
- A led showing the current state of the connection

3.3.2 How to run it

There are several way to launch this GUI. The first one, assuming the correct dependencies are installed (tkinter, pillow, python dev lib, ...), simply run the python script this way:

```
python3.4 ui.py
```

However, if you don't want to install all those python decencies, you can run the compiled python script[3]. Assuming you have the required graphical and serial packages installed on you computer.

```
./ui
```

[3]Using pyinstaller (it bundles a Python application and all its dependencies into a single package)

Otherwise, if you just want to verify that the basic feature is working properly, you can simply run the read_usb script that only need the serial python library. This script simply capture the input over the usb port and simulate the keyboard's arrow_buttons press.

```
./read_usb.py
```

Finally, if you want to try our nice GUI without installing, all dependencies and packages, you can ask one member of the group[4] to send you a virtual machine image with everything installed.

4 Difficulties

First of all, we had quite a hard time to compile and upload on the Re-mote. This was due to the fact that we didn't know that we had to update the contiki folder with `git pull`. However, even with the folder updated, uploading on the Re-mote frequently throws an error.

Secondly, we found out that the battery sensor and the phidget sensor for the joystick don't work well together. As soon as the battery sensor is activated on the Z1, the joystick sensor stays with its default values whatever we do. Because the battery sensor is used rarely, we decided to simply keep the battery sensor deactivated. Every 5 min, when we need to check the battery, we first deactivate the phidget sensor before activating the battery sensor. After reading the value of the battery sensor, we deactivate it and then reactivate the phidget sensor.

5 Wired versus Wireless reaction

The figure 2 shows a comparison of the number of received inputs during 5 seconds between the wireless and the wired mode. The distance between the Z1 and the Re-mote was progressively increased. The goal of this measurement was to evaluate at which maximum distance can the player be from its computer without suffer too many delay. The results are unexpected as you will see. To realize those measurements, we first put the Re-mote and the Z1 side by side and launch the python script **statistics_input_received.py** that count the number of received inputs on the serial USB interface during 5 seconds. For each distance, we repeated the measurements 3 times. Then, we moved the Z1 10 centimeters further from the Re-mote and start again. Finally, we did an average of our results. We also launched the script with the Z1 directly plugged to the computer to have an idea of the maximum rate that can be reached.

There is a very surprising decrease between 48 centimeters and 50 centimeters. We repeated our measurements multiple times and the number of received inputs always drop from 30 to 5 when we moved from 48 cm to 50 cm. This is unexpected. We expected to see a linear or an exponential decrease, but not a constant value followed by a sudden drop and then again a more or less constant value. Finally, reaching 2 meters of distance, no more inputs are received by the Re-mote.

We didn't find any explanation for the sudden drop that arise between 48 and 50 cm. At first, we were measuring every 10 centimeters but this drop has aroused our curiosity so we make some measurement closer to 50 cm and realize that the drop arise when we were moving the Z1 from 2 centimeters.

[4]sami.mokaddem@student.uclouvain.be

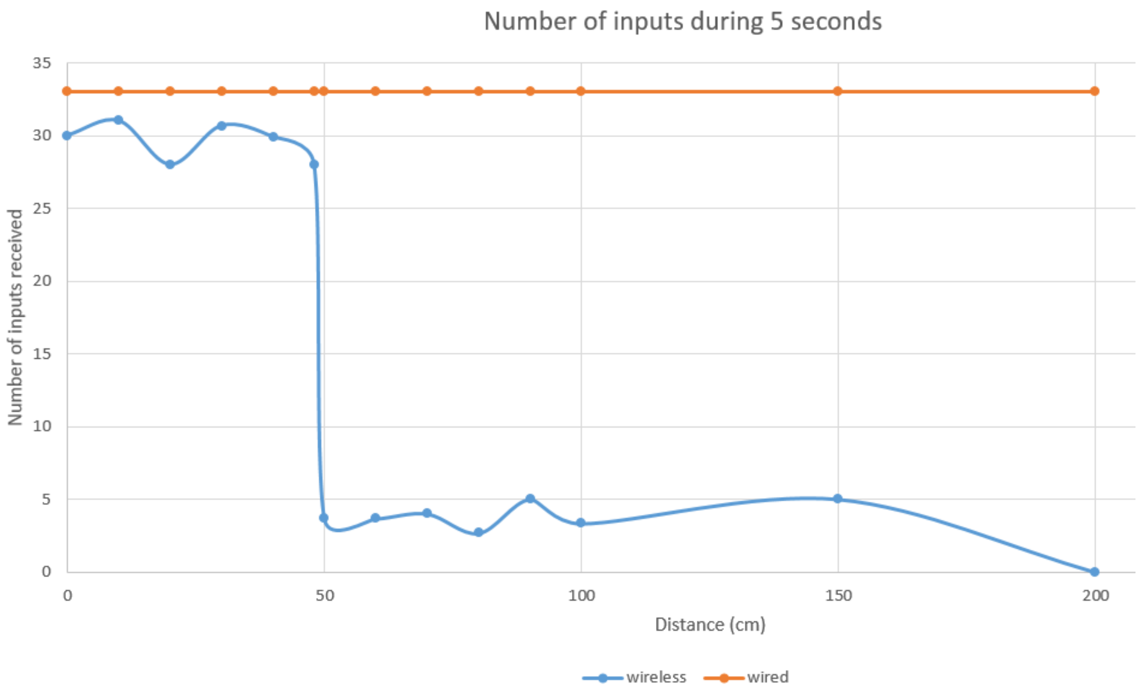

Figure 2: Comparison of number of inputs received during 5 seconds

On the other side, it is comfortable to see that the wired measurements always show the same value.

6 Conclusion

This project was really fun to work on. It increased quite a lot our interest about this large field which is the **Internet of Things**. We thought that it would be much harder to develop for devices such as the Z1 and the Re-mote (and we were a little afraid to work with the C language).

We are aware that the devices we used are not designed for real-time communication but much more designed for low-power mode communication. So we think our idea of a game controller would work better on a more suited hardware.

Some improvements could be made:

- Not using broadcast but instead unicast, that would allow to have multiple controllers connected to the same Re-mote.
- Instead of having absolute direction without a scale (Top, Bottom, ...), we could have implemented a scalable way to interpret the data of the Joystick (0.5 top - 0.3 right).
- Using the Z axis of the accelerometer to add more possibilities of movements.
- Implement a solution where, instead of sending every 20 ms all the directions, only send new directions when they occur or send a message to withdraw a direction when it stops. This improvement would give the possibility to simulate a long press of a keyboard input instead of repetitively ”hitting” the touch again. But this can raise other concerns like packet lost and so on.