LINCOLN UNIVERSITY & UNIVERSITÉ DE LA RÉUNION

Robotable Project

Author:
Arnaud CHEN-YEN-SU

Supervisors:
Dr. Stuart CHARTERS
Dr. Keith UNSWORTH

August 26, 2013

Abstract

In 2002, Lincoln University, in collaboration with Tufts University, started to develop a Robotable environment to enhance learning about robotics and encourage engineering problem solving.

A Robotable is a tabletop that acts as a rear-projection screen. A picture is projected on it using a mirror (45 degrees inclined) and a projector. The system is completed by an optical tracking system used to detect interaction on the tabletop.

In 2012, Leshi Chen, a master student at Lincoln University, worked to improve the Robotable environment. His goal was to facilitate the use of Robotables by providing a set of toolkits (in C#) for setting-up and creating easily new games.

My project was to port Leshi's toolkits to a more portable computer: a Raspberry Pi. With a Raspberry Pi, setting-up Robotables environment will be easier thanks to its low price and small size. In fact, rather than using a standard computer (with a monitor, keyboard. . .) we will be able to use a computer the size of a credit card.

The easiest way to port the toolkits was to rewrite them in another programming language: Python. Toolkits' architecture design and the algorithm remain almost the same. The main differences are that my software does not provide the same level of detail and the Network toolkit uses a different architecture.

At the end, the software in Python provides almost the same result as the initial software. Nonetheless, improvements still need to be made, specially concerning the Network and Game Management toolkit.

Résumé

En 2002, Lincoln University, avec la collaboration de Tufts University, ont commencé à développer le concept de "Robotable environment" afin d'améliorer l'apprentissage des technologies en rapport avec la robotique et les sciences de l'ingénieur.

Une Robotable est une table transparente qui sert d'écran de projection. La projection est effectuée en utilisant un miroir (incliné à 45 degrés) placé en dessous de la Robotable. Le système est complété par un mécanisme qui permet de détecter les interactions sur la Robotable.

En 2012, Leshi Chen, étudiant en Master à Lincoln University, a travaillé à l'amélioration du système de Robotable. Son but était de faciliter l'usage des Robotables en fournissant un ensemble de programme (en C#) qui permettrait la configuration et création rapide de nouveaux jeux.

Mon projet a été de porter le programme sur un Raspberry Pi. Avec le Raspberry Pi, la mise en place de "Robotables environment" est plus facile grâce à son faible coût et à sa petite taille. En effet, au lieu d'utiliser un ordinateur standard (avec unité centrale, écran. . .), on peut simplement utiliser le Raspberry Pi qui a la taille d'une carte de crédit.

La méthode la plus simple pour porter le programme a été de le réécrire en utilisant un autre langage de programmation : Python. L'architecture du programme et son mode de fonctionnement sont restés quasiment identiques. Les principales différences sont que le nouveau programme ne possède pas le même niveau de détails que l'original et l'architecture de la partie qui gère les échanges de données entre les Robotables est différente.

Au final, le programme en Python fournit quasiment le même résultat que le programme original. Cependant, des améliorations restent encore à faire, particulièrement concernant la partie réseau et gestion du jeu.

Acknowledgements

I would like to acknowledge my supervisor team Stuart Charters and Keith Unsworth for their support and direction. They gave me the great opportunity to improve my skills in software engineering and in English. They also helped me to discover a new culture.

Thank you to the team at the ESIROI for their help for the application process and trip organisation.

Finally, I would like to thank all who have helped me to do this internship.

Contents

List of Figures

Chapter 1

Introduction

I did my internship of three months at Lincoln University, New Zealand. My supervisors were Dr. Stuart Charters (Head of Department) and Dr. Keith Unsworth (Postgraduate Convenor). My work was to port some existing toolkits written by a Master student in C# to a Raspberry Pi.

At the end of my internship, the Raspberry Pi should be able to use a Wiimote Controller using a bluetooth USB dongle, and therefore, run some existing games on the RoboTable. Extensions to my project were to build new games and a web based control mechanism to ease the configuration and launch of games.

1.1 Structure of the Report

I will first present the University where I did my internship and persons who worked with me. Chapter 2 presents the initial project and explains some technical terms used. It also explains my work in depth and the final result. In the last part, I conclude on what I learned during this internship and future extensions of this project.

Appendix A explains how to use the software for an user. Appendix B provides useful information for a developer. Appendix C provides the complete API Documentation.

Chapter 2

Presentation of the University

2.1 History

Lincoln University was created in 1878 as a School of Agriculture and begins received students in 1880. From 1896 to 1961 it served students under the name "Canterbury Agricultural College", and offered qualifications of the University of New Zealand until that institution's demise. From 1961 to 1990, it was known as Lincoln College, a constituent college of the University of Canterbury, until achieving autonomy in 1990 as Lincoln University. It is the oldest agricultural teaching institution in the Southern Hemisphere.

2.2 Faculties & Departments

Lincoln University is composed of three faculties:

- Agriculture and Life Sciences
- Commerce
- Environment, Society and Design

Each faculties are composed by departments. I did my internship in the Environment, Society and Design's faculty, in the department of **Applied Computing**. This department is involved in research and, undergraduate and postgraduate teaching. Current interests include: Visualisation, Computer Graphics, Image Processing, Data modelling and management, Simulation and Modelling, and Computing and Education. There are currently nine persons working in the Department of Applied Computing. The Head of Department is Dr. Stuart Charters (also my supervisor).

Chapter 3

Current system

3.1 History

In 2002, Lincoln University, in collaboration with Tufts University, USA, started to develop a Robotable environment to enhance learning about robotics and develop engineering problem solving in school children. The first prototype of Robotable environment was created in 2005 at Lincoln University by Paul S. Mason in his thesis "A Hands-On Tabletop Environment to Support Engineering Education".

3.2 What's a Robotable?

A Robotable is a tabletop that acts as a rear-projection screen (Mason [2005]). A picture is projected on to it using a mirror (45 degrees inclined) and a projector. The system is completed by a tracking system used to detect interaction on the tabletop.

The first tracking system used patterns to detect those interactions, but was later improve using a Wimote Controller that can track Infrared LEDs (up to four). LEDs are attached on a board below the LEGO Mindstorm (Figure 3.1).

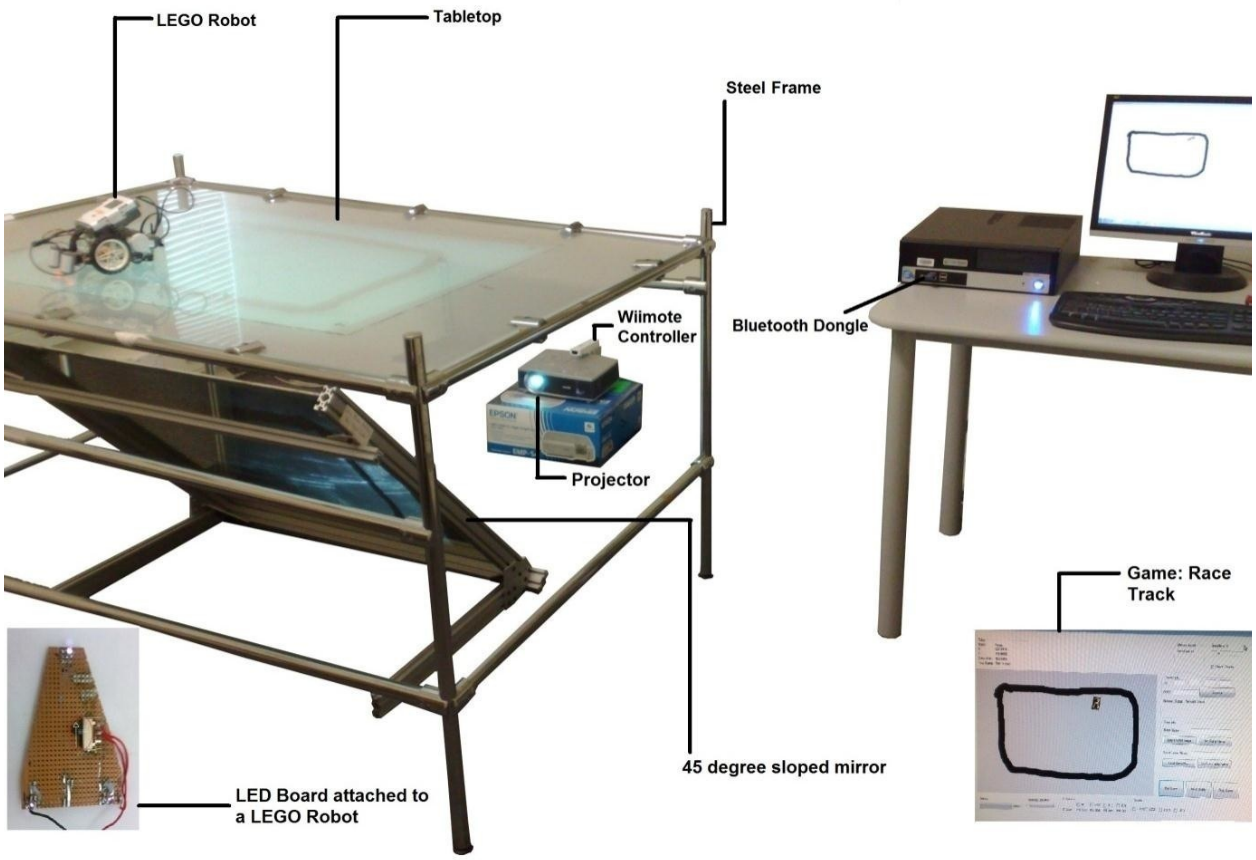


Figure 3.1: Robotable environment, source: Chen [2012, p. 7]

3.3 Presentation

In 2012, Leshi Chen, a master student at Lincoln University, worked to improve the Robotable environment. His goal was to facilitate the use of Robotables by providing a set of toolkits (in C#) for setting-up and creating easily new games (Chen [2012]).

3.4 Toolkits architecture

His software is separated into four different toolkits :

- Communication toolkit
- Game Management toolkit
- Network toolkit
- Robot Tracking toolkit

The figure (Figure 3.2) below explains how the software works using the toolkits.

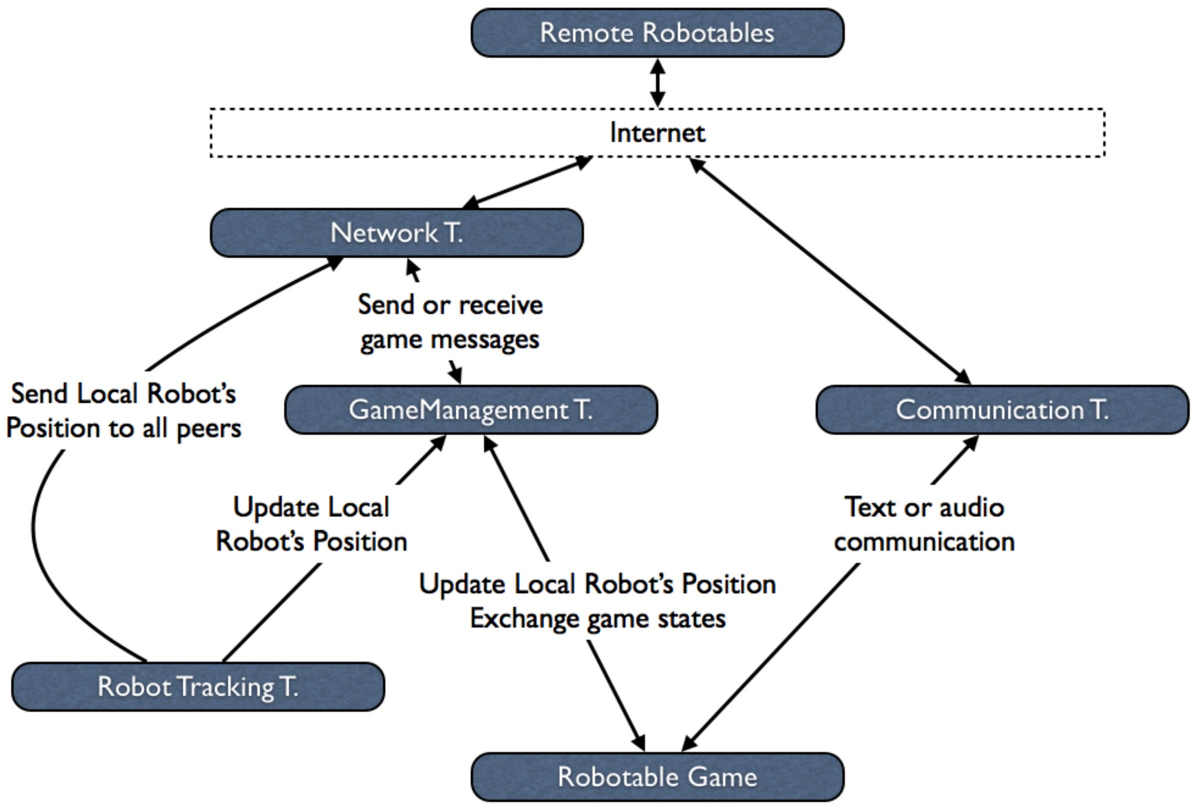

Remote Robotables
Internet
Network T.
Send or receive game messages
Send Local Robot's Position to all peers
GameManagement T.
Communication T.
Update Local Robot's Position
Text or audio communication
Update Local Robot's Position Exchange game states
Robot Tracking T.
Robotable Game

Figure 3.2: Toolkits architecture

Chapter 4

My Project

4.1 Subject

My project was to port Leshi's toolkits to a more portable computer : the Raspberry Pi. An extension to this, was to design and implement a web based control mechanism for starting games and/or create new games.

4.2 Raspberry Pi

What's a Raspberry Pi?

A Raspberry Pi is a small computer (size of a credit card) that acts as a standard computer[1]. Thanks to its low price (around USD 25 and 35), it's attractive for educational work (research, learn programming. . .) as well as personal usage (word-processing, media-center. . .). The default distribution is a modified version of Debian.

Concerning the hardware specifications[2], the Raspberry Pi's CPU is a 700 MHz ARM11 and it has a memory (SDRAM) of 256MiB or 512MiB (depending on the model). The GPU is capable of BluRay quality playback.

Why to choose a Raspberry Pi?

We wanted to replace the standard computer (Figure 3.1) by a Raspberry Pi. In fact, it will be easier for people who want to create Robotables environment to buy a Raspberry Pi rather than a standard computer with a monitor, mouse and keyboard. Moreover, it will be also easier to provide a SD Card with all softwares needed for using a Robotable.

[1]http://www.raspberrypi.org/faqs
[2]http://elinux.org/RPi_Hardware

Figure 4.1: Raspberry Pi, source: `http://en.wikipedia.org/wiki/File:RaspberryPi.jpg`

4.3 First idea

The toolkits were written in C#, and therefore, more specific to Windows operating system. The first idea was to try to port the toolkit to the Raspberry Pi using MonoDevelop[3].

MonoDevelop is a cross-platform IDE for C# and .NET languages. It provides a way to execute software written in .NET on Linux and Mac OSX operating systems.

However, we had two main issues with MonoDevelop. The first issue that we had was that the initial library used for the bluetooth USB dongle and Wiimote Controller was not working using MonoDevelop. Another issue was that the initial library used was DirectX for the graphical part and this does not work with MonoDevelop.

Moreover, Leshi didn't write complete documentation for a developer who wants to port the software. In the end, even if we had a lot of code to rewrite, it seemed still easier and faster to rewrite the complete software.

4.4 Python

We decided that we will rewrite the software in Python. In fact, Python is a cross-platform programming language and known to be easy to read and write. It provides also a lot of useful libraries, including a library to work with Wiimote Controllers and GUI. Moreover, it was a good opportunity for me to learn a new programming language.

[3]http://monodevelop.com/

4.5 New Architecture

The new architecture in python works almost the same way as the previous architecture (Figure 3.2). The figure below (Figure 4.2) explains how the scripts in Python work together.

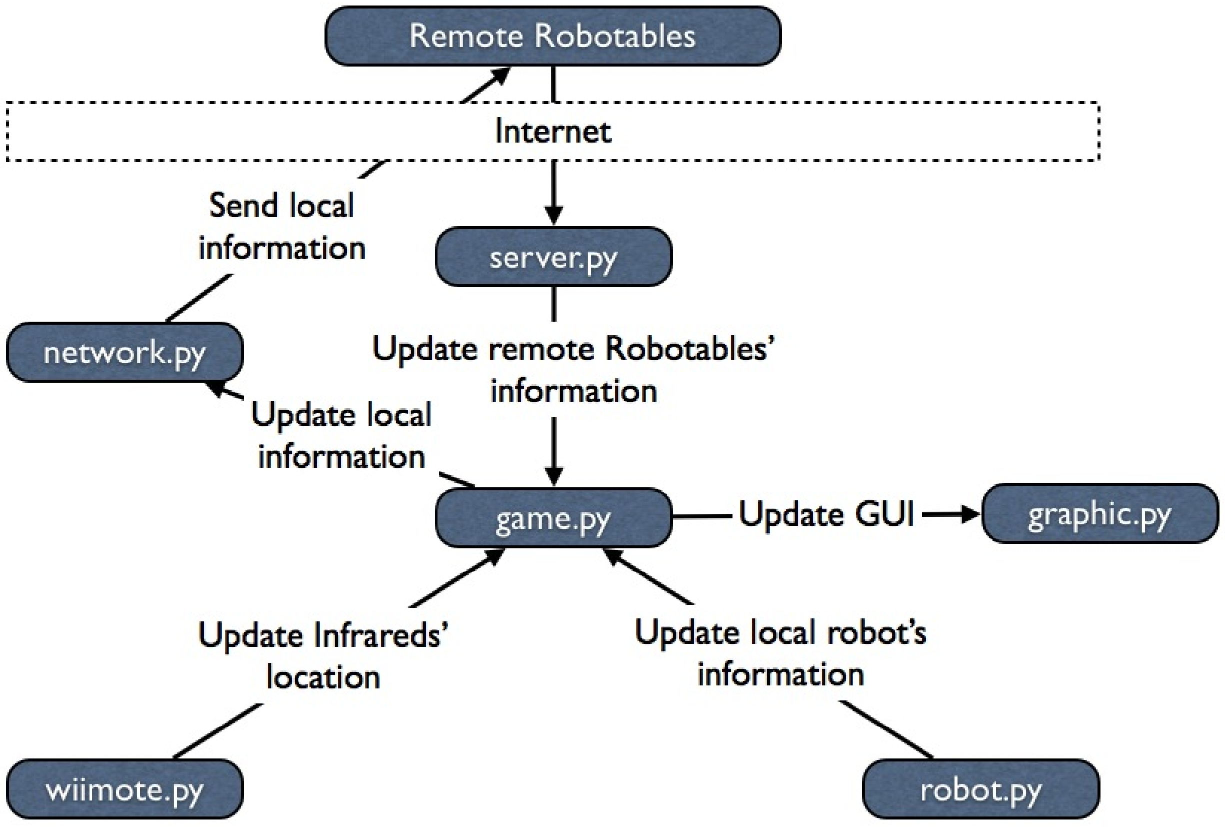


Figure 4.2: New Architecture

4.6 Robot Tracking toolkit

The new robot tracking toolkit corresponds to the script wiimote.py and robot.py. The script wiimote.py returns information about infrareds that are detected and robot.py returns information about robot on the robotable. The library used for the wiimote controller is cwiid[4]. It's written in Python and even if last modifications was done in 2010, this library provides everything needed for the project.

4.7 Game Management toolkit

The game management toolkit is represented by the script game.py. Within this script, there are two classes: Game and GameManagement. At the moment, the GameManagement class executes everything that's need to be done before the launch of a game (do calibration, send IP address…). The Game class starts the game and manage the progress of the game.

[4]https://github.com/abstrakraft/cwiid

4.8 Network toolkit

Leshi wrote the network toolkit using sockets. We thought that it'll be easier for developers to work with this toolkit if it was written using HTTP protocol. On Figure 4.3 you can see the new architecture of the network. The new architecture is called REST(**R**epresentational **S**tate **T**ransfer) Architecture and mainly characterised by his Client-Server and Stateless architecture (more information can be found on Wikipedia[5]).

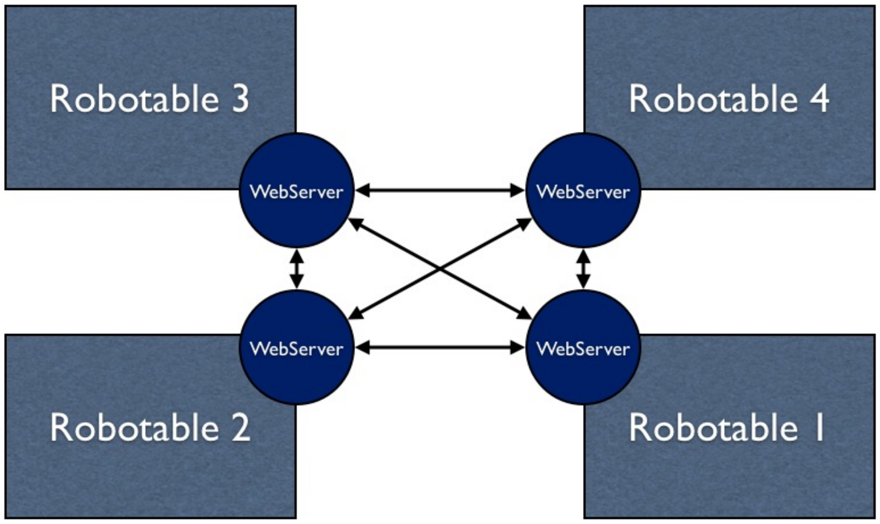


Figure 4.3: New network architecture

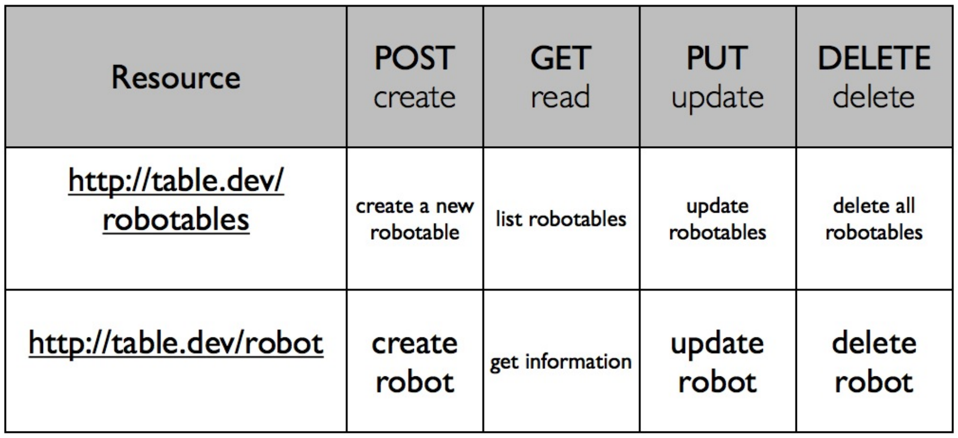
RESTful API

With this new architecture, we started to implement a RESTful API (Figure 4.4). However, I didn't have the time to finish entirely this API.

Web interface

With a web interface, teachers will be able to control/setup robotables easily using for example tablets, phones or laptops. I thought that it will be easier to write the web interface and the network toolkit at the same time. In fact, the script server.py provides the RESTful API and the web interface. At the moment, the web interface provides only a way to start and stop the game.

[5]http://en.wikipedia.org/wiki/Representational_state_transfer



Resource	POST create	GET read	PUT update	DELETE delete
http://table.dev/robotables	create a new robotable	list robotables	update robotables	delete all robotables
http://table.dev/robot	create robot	get information	update robot	delete robot

Figure 4.4: RESTful API

4.9 Performance

Simple tests were perform to know the performance of the new software. We simulate a game with three robotables. The Raspberry Pi's percentage of CPU used during the game was around 60%. Nonetheless, use of HTTP protocol seems to be a bit slow: delay between responses was around 250ms.

4.10 Tools

Django & Flask

Django[6] is a Python Web framework. I started to use it for the network toolkit and the web interface, but I found it to complex for the simple usage of this project. That's why I used Flask[7] instead. Flask is a microframework for Python, and it's easier to use than Django.

JSON

I needed a way to exchange data between Robotables. JSON[8] was a simple solution.

Sphinx

Leshi's code was hard to understand even with the documentation that he wrote. Therefore, I wanted a simple way to write a better documentation. Sphinx[9] is a tool that provides you a way to write the documentation at almost the same time as your code. Moreover, it provides several outputs format like HTML, Latex or PDF. You can see the output format in PDF in Appendices (section 5.4).

Github

Github[10] is a web application based on the software Git[11]. Git is a free and open source distributed version control system. With Github, I was able to keep track of the different versions of my software and keep a backup online[12].

[6]https://www.djangoproject.com/
[7]http://flask.pocoo.org/
[8]http://www.json.org/
[9]http://sphinx-doc.org/
[10]https://github.com/
[11]http://git-scm.com/
[12]https://github.com/arnaudchenyensu/RoboTableProject

Chapter 5

Summary

5.1 Difficulties

The first difficulty that I had was to learn Python. In fact, I only used Python once and therefore, I had to learn the basics, good practices and Object Oriented Programming in Python. Another difficulty that I had was the calibration of the Robotable.

5.2 Final Result

The software that I wrote provides almost the same result as Leshi's software but not with the same level of details. A concrete example is that Leshi's software provides a system of skeleton project and game rules that makes easier the creation of new games. With my software, you will have to write more code.

In addition to the main software, a simple web interface for starting games was created and the documentation for user and developer was written.

5.3 Future

Even if the new software works on a Raspberry Pi, improvements still need to be made. These improvements mainly concern the network toolkit and the game management toolkit. Moreover, the web interface does not provide a way to configure games and the GUI's performance are actually link with X11 server. X11 server does not use the capacity of Raspberry Pi's GPU and therefore performances are bad. However, the Raspberry Pi foundation planned to replace X11 by Wayland, which uses GPU. All of these improvements should be part of the future of the project.

5.4 Conclusion

Thanks to this internship, I had the great opportunity to improve my skills in programming and English. I was also able to discover a new country and culture, and meet new people.

Appendices

In 2002, Lincoln University, with the collaboration of Tufts University, started to develop a Robotable environment to enhance learning about robotics and engineering problem solving.

A Robotable is a tabletop that acts as a rear-projection screen. A picture is projected on it using a mirror (45 degrees inclined) and a projector. The system is completed by an optical tracking system used to detect interaction on the tabletop.

In 2012, Leshi Chen, a master student at Lincoln University, worked to improve the Robotable environment. His goal was to facilitate the use of Robotables by providing a set of toolkits (in C#) for setting-up and creating easily new games.

My project was to port Leshi's toolkits to a more portable computer: a Raspberry Pi. With a Raspberry Pi, setting-up Robotables environment will be easier thanks to its low price and small size. In fact, rather than using a standard computer (with a monitor, keyboard...) we will be able to use a computer with the size of a credit card.

The easiest way to port the toolkits was to rewrite them in another programming language: Python.

This documentation explains how to use the software on a Raspberry Pi.

Appendix A

User Guide

This part of the documentation explains how the software works and how to run the game example on your own robot table.

A.1 Introduction

What's a Robot Table?

A Robotable is a tabletop that acts as a rear-projection screen (Mason [2005]). A picture is projected on to it using a mirror (45 degrees inclined) and a projector. The system is completed by a tracking system used to detect interaction on the tabletop.

The first tracking system used patterns to detect those interactions, but was later improve using a Wimote Controller that can track Infrared LEDs (up to four). LEDs are attached on a board below the LEGO Mindstorm (Figure A.1).

Requirements

Hardware

- Robot Table
- Projector
- Rasberry Pi
- Wiimote
- Dongle USB Bluetooth

Software

- cwiid
- Tkinter
- Flask
- Requests
- ImageTk

More information on how to install these software can be found *here*.

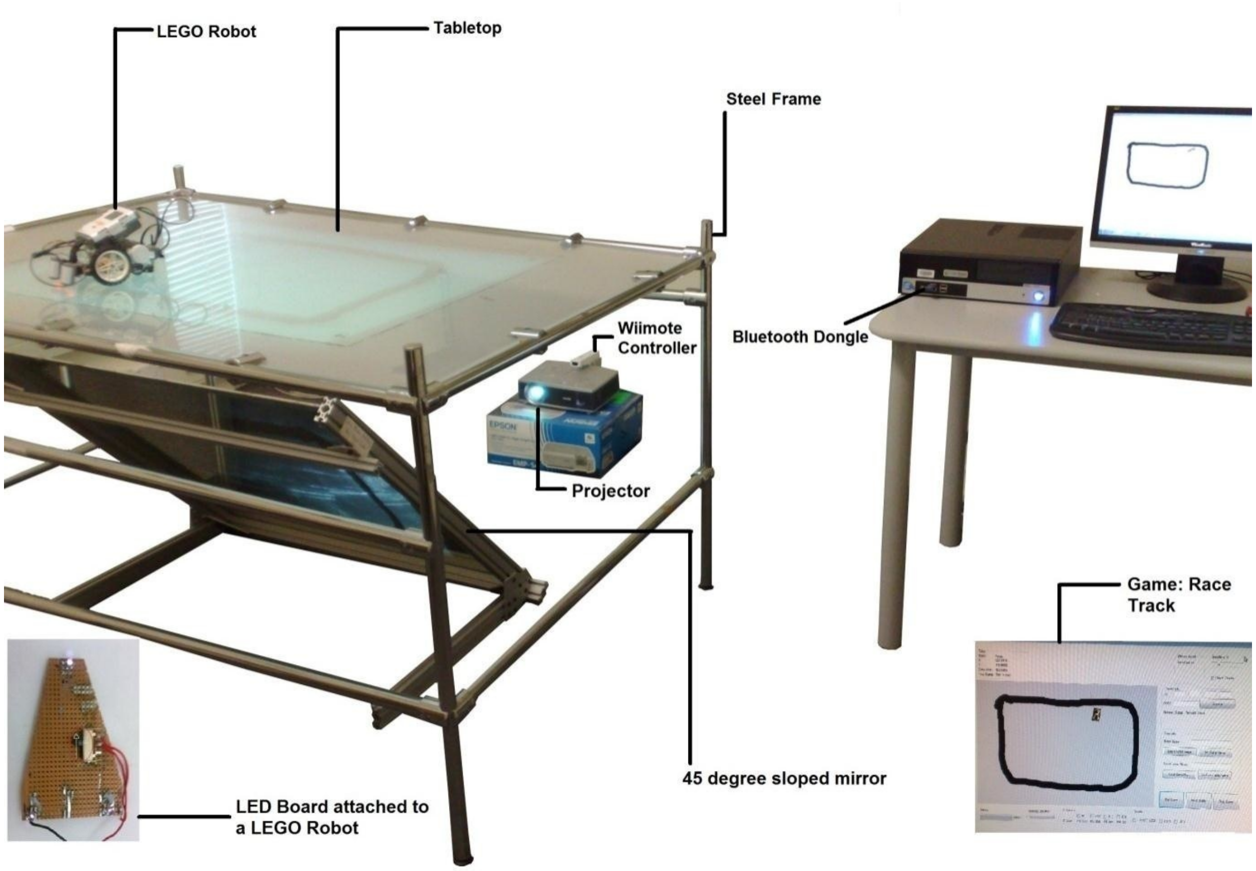


Figure A.1: Robotable environment, source: Chen [2012, p. 7]

A.2 Installation

Get the Code

The code is available on Github. You can easily clone the repository:

```
$ git clone https://github.com/arnaudchenyensu/RoboTableProject.git
```

Installing the requirements

The library cwiid, ImageTk and for the dongle USB bluetooth can be installed using apt-get:

```
$ sudo apt-get install python-cwiid python-imaging python-imaging-tk bluetooth bluez-utils blueman
```

The other softwares can usually be installed using pip or easy_install.
You will find more information on each software's website.

A.3 Quickstart

When every *requirements* are met, you'll need to launch the X server:

```
$ startx
$ export DISPLAY=:0.0
```

Then, you can launch the server using the script server.py:

```
$ python server.py
```

The web server is available at the Raspberry Pi's IP address on the port 5000. If the ip address of your Raspberry is 10.4.9.4, you can access the server at http://10.4.9.4:5000. You can then launch your game using the button 'Start game'.

Appendix B

Developer Guide

This part of the documentation explains how to write a new game using the existing scripts.

B.1 How to create a new game?

If you want to create a new game, you will have to create a new class that inherits from the Game class. Then you need to override the method start. When you override this method, you'll need first to call these methods:

```
self.game_management.start(self.addr_main_server, self.is_main_server(), self.addr)
self.x_factors = self.game_management.x_factors
self.y_factors = self.game_management.y_factors
self.servers = self.game_management.servers
```

You then write every thing you want to execute for your game and finish by these lines:

```
self.gui.after(100, self._update_map)
self.gui.mainloop()
```

In the example above, the method _update_map will be call every 100ms, and therefore, you will have to override this method (for updating your drawings, check state of the game...).

Appendix C

API Documentation

This part of the documentation details every function, class or method used for the Robot Table Project.

C.1 API Documentation

Game

class `game.Game`(*robot*, *sensor*, *network*, *gui*, *game_management*, *addr_main_server*)

Create a game.

Parameters

- **robot** – Robot object used on the table.
- **sensor** – Sensor object used to detect IRs.
- **network** – Network object used to communicate between tables.
- **gui** – GUI object.
- **game_management** – GameManagement object.
- **addr_main_server** – Main server's IP address.

`draw_robots()`

Draw robots on the table.

`get_addr()`

Return the ip address of the server.

Note: the solution was find on Stackoverflow.

`is_main_server()`

Return if the server is the main server.

`load_map`(*path*)

Load the map on the canvas.

`start()`

Launch the game.

Note: This is the method to override when creating your own game.

`stop()`

Stop the game.

class `game.GameManagement`(*sensor*, *gui*, *network*, *nb_servers*)

docstring for GameManagement.

Parameters

- **sensor** – Sensor object used to detect IRs.
- **gui** – GUI object.
- **network** – Network object used to communicate between tables.

- **nb_servers** – Number of player/server.

`do_calibration()`
Draw 5 crosshairs and then return the calibration's factors (x_factors and y_factors) using the _calculate_calibration_n method.

`is_servers_ready()`
Return true if all servers are ready.

`launch_game()`
Launch the game on each server.

`send_list_servers()`
Send a list containing the address of all servers to each server.

`start(`*addr_main_server, is_main_server, addr*`)`
Execute the needed steps before launching a game.

Parameters

- **addr_main_server** – Main server's IP address.
- **is_main_server** – True if this is the main server.
- **addr** – address of the server on which this method is called.

`synchronise_servers()`
Synchronise each servers and launch the game.

`wait_addr_servers()`
Wait that all servers have sent their address.

`wait_servers()`
Wait that all servers are ready.

Graphic

class `graphic.GUI`
docstring for GUI

`after(`*secs, function*`)`
Call the function every secs.

Parameters

- **secs** – Time interval in seconds.
- **function** – The function to call.

`coords(`*object_id, *coords*`)`
Change and return the coordinates of the object.

Parameters

- **object_id** – Id of the object.
- ***coords** – (optional) List of coordinate pairs.

Note: Same effect as the Tkinter's method.

`delete(`*object_id*`)`
Delete the object with the id object_id.

Parameters **object_id** – Id's object.

`init_graphic()`
Initialize the graphical part using Tkinter.

`load_map(`*path*`)`
Load the map.

Parameters **path** – Path to the map.

`mainloop()`
Start the graphic mainloop.

`set_full_screen()`
Set the window in full screen.

class `graphic.Crosshair`(*gui, x, y, rad=30, color='red', width=2*)
Create a Crosshair object.

Parameters
- **gui** – GUI object.
- **x** – x location of the Crosshair.
- **y** – y location of the Crosshair.
- **rad** – (optional) Crosshair's radius.
- **color** – (optional) Crosshair's color.
- **width** – (optional) Width's lines.

`delete()`
Delete the crosshair on the canvas.

`draw()`
Draw the Crosshair on the screen.

class `graphic.RobotDrawing`(*gui, rad=10, outline_color='red', fill_color='green'*)
Create a RobotDrawing object, represented by 3 dots on the screen.

Parameters
- **gui** – GUI object.
- **rad** – (optional) Radius of the dots.
- **outline_color** – (optional) Outline color of the dots.
- **fill_color** – (optional) Fill color of the dots.

`draw`(*leds*)
Draw three dots on the screen.

Network

class `network.Network`(*port=5000*)
Create a Network object.

Parameters **port** – (optional) Port used to communicate.

`format`(*addr_server*)
Format the address and return it.

`get_irs`(*addr_server*)
Return the IRs' location of a server.

`is_ready`(*addr_server*)
Return True if the server is ready.

`launch`(*addr_server*)
Launch the game on the server.

`send_addr`(*addr_server, addr_to_send*)
Send an address to a server and return the response.

Parameters
- **addr_server** – the address to send to.
- **addr_to_send** – the address to send.

`send_list_servers`(*addr_server, list_servers*)
Send a list of servers to a server.

Robot

class `robot.Robot`(*sensor*, *gui=None*)

This class represent a Robot with 3 leds (2 at the back and 1 at the front).

Parameters

- **sensor** – Sensor that detect the infrared.
- **robot_drawing** – (optional) RobotDrawing object.

`centre`

Return a tuple with X and Y location of the robot's centre

`leds`

When this property is called, leds position is automatically updated and returned.

Usage:

```
>>> r = Robot(Sensor())
>>> r.leds
{'front': {'X': 10, 'Y': 20}, 'left': {'X': 103, 'Y': 23}, 'right': {'X': 111, 'Y': 203}}
```

Note: Since leds position is automatically updated at every call, you should save leds location in a variable (e.g leds = robot.leds)

Wiimote

class `wiimote.Wiimote`(*width_resolution=1024*, *height_resolution=768*, *test=False*)

Create a Wiimote object and connect to the dongle USB.

Parameters

- **width_resolution** – (optional) Width resolution of the sensor
- **height_resolution** – (optional) Height resolution of the sensor
- **test** – (optional) Only use for test-driven

`connect()`

Connect the wiimote to the dongle Bluetooth.

`disconnect()`

Disconnect the wiimote to the dongle Bluetooth.

`get_leds()`

Return a list of the location of leds detected.

(e.g [{'X': 10, 'Y': 20}, {'X': 103, 'Y': 23}, {'X': 111, 'Y': 203}, {'X': 121, 'Y': 13}])

Note: If the location of a led is not detected, X and Y equal -1.

Index

Bibliography

Leshi Chen. Supporting distributed multiplayer robotable games. Master's thesis, Lincoln University, 2012. URL `http://hdl.handle.net/10182/5253`. v, 4, 15

Paul S. Mason. A hands-on tabletop environment to support engineering education. Master's thesis, TUFTS UNIVERSITY, 2005. 3, 14