Notice - Instructions
Manticore project

Shanghai Jiao Tong University
Wireless Sensor Network Laboratory

Thibault FORDEVEAUX
Shilong JIANG
Ghita TAZI

July 1st, 2015

1 - Description of Manticore

In this section, we will present you succinctly the Manticore project and describe our deliverable.

1.1 The Manticore project

Manticore is a cross-platform distributed overlay networking framework for smart devices. Based on it, devices such as smartphones, can discover the existence and capability of each other as well as establish publishing/subscribing service mutually.

Manticore counts four elements: core, client, data-provider and end-point.

All data transmission is carried by the OSC data flow standard.

The artist described his idea as followed: first, comedians are going to use smartphones to simulate the fact that they want to control the environment that surround them, which means for example change the music and generate light effects. They will not succeed to do that. Then, the main actor is going to intervene and try it in turn, launch an application on his mobile device and succeed to control the music and lights.

To manage to do that, the Manticore project is planned to be used for this part of the Concert.

The idea is to set up in the Concert room a network composed of four nodes and connect to each node two smartphones. In each smartphone, a mobile application, which is part of the Manticore project, is going to be installed. Every time an actor starts the application, while connected to a specific WiFi, the smartphone starts sending raw accelerometer data to the node its attached to, thanks to the wireless network. The data is first received by a data-provider within a WebSocket, then sent to the client's end-point using a UDP socket. For the concert, the client is a Max/MSP interface. It is going to process the incoming raw accelerometer data and transform it into a sound. Every time an actor is going to move its smartphone, it will produce a certain sound so that the audience will have the illusion that the music is controlled by the actor itself.

1.2 Hardware part

To use the Manticore project within the Concert at Shanghai Conservatory of Music, two types of hardware are needed: Cubieboards and smartphones.

Cubieboard is a single-board computer, made in Zhuhai, Guangdong, China. The first short run of prototype boards were sold internationally in September 2012, and the production version started to be sold in October 2012. It can run Android 4 ICS, Ubuntu 12.04 desktop, Fedora 19 ARM Remix desktop, Archilinux ARM or a basic Debian server via the Cubian distribution.

It uses the AllWinner A10 SoC, popular on cheap tablets, phones and media PCs. This SoC is used by developers of the lima driver, an open source driver for the ARM Mali GPU.It was able, at the 2013 FOSDEM demo to runioquake 3 at 47 fps in 1024×600.

Our main device is the Cubieboard4 also known as CC-A80. It is based on an Allwinner A80 SoC (quad Cortex-A15, quad Cortex-A7 big.LITTLE), thereby replacing the Mali GPU with a PowerVU GPU. The board was officially released on 10 March 2015.

- SoC: Allwinner A80
- CPU: 4x Cortex-A15 and 4x Cortex-A7 implementing ARM big.LITTLE
- GPU: PowerVR G6230 (Rogue)
- Video acceleration: A new generation of display engine that supports H.265, 4K resolution codec and 3-screen simultaneous output
- MicroUSB 3.0 OTG
- Connectivity – Gigabit Ethernet, Wi-Fi 802.11 b/g/n + Bluetooth 4.0 (AP6330 module)
- Operating System : Linaro-server-v1.0 - Linaro-desktop-v4.0

This device will provide a private wireless network using the WPA-PSK mechanism which refers to Pre-shared key mode. This is designed for home and small office networks and doesn't require an authentication server. Each wireless network device authenticates with the access point using the same 256-bit key generated from a password or passphrase.

Once plugged in, the Cubieboard starts acting as a WiFi Hotspot, by creating a WiFi network. It starts automatically Manticore and detects all the sensors connected to it; as well as the nodes and their sensors present in the network.

In total, the network sets up for the August Concert will be composed of four Cubieboards. Each Cubieboard will be named following the pattern: Cubieboard_#X.

To each one of the four Cubieboards, two smartphones will be connected as sensing devices. The smartphones use Android as their operating system.

1.3 The Manticore project

1.3.1 Manticore Network

In this part, we are going to present the architecture that we will deliver you and it's described by the following figure:

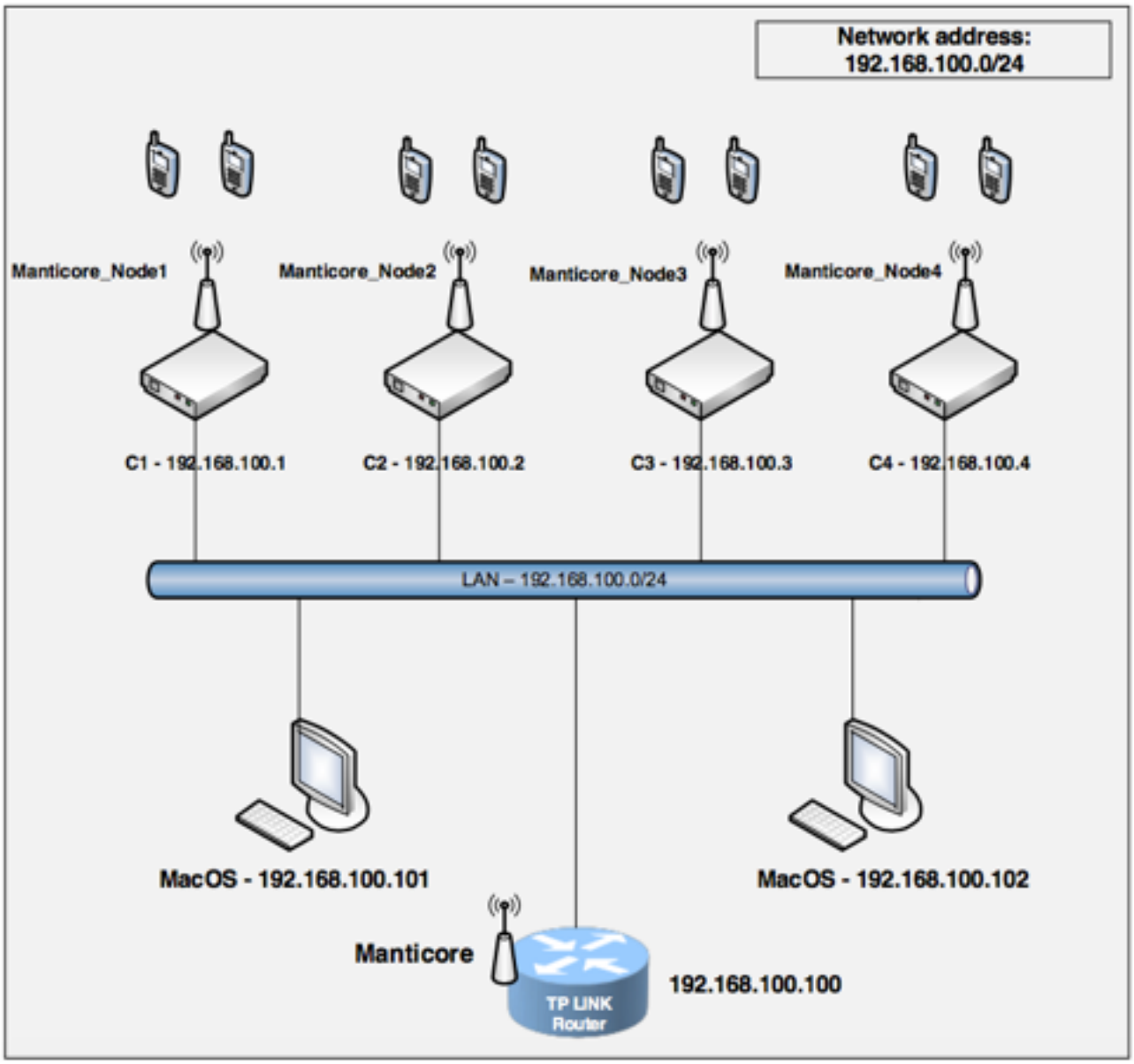


Figure 1 - Architecture of the Manticore Project

The whole network is using the IP address 192.168.100.0/24.

Each Cubieboard has a static IP address like describe in the previous figure 192.168.100.X and on each Cubieboard is connected smartphones through the access point created by the device. Cubieboards are connecting to the router by ethernet cable.

Each smartphone connect randomly to the Cubieboard depending on the wireless signal strength.

Each computer connected to the network get its IP address by the DHCP server implemented into the TP-Link router (from 192.168.100.101 to 192.168.100.254) and are connected wirelessly to the network *NovalMusic* (name of the TP-LINK's access point).

Nota: The password of *NovalMusic* network is: 912345678

Nota 2: If you want to use an other network, you have to change every configuration file of each Cubieboard

1.3.2 Mobile application

The mobile is acting as a sensing device in the Manticore system.

Accelerometer data is collected every time the user orientates its device in a certain direction, and sent to the Cubieboard, in charge of forwarding it to the final client (the smartphone is sending OSC packet to the Cubieboard).

Within the framework of the concert, we were asked to set up a user interface with activity and moving objects in the space, to make sure the audience understands the power of the mobile device and its role in controlling light and sound effects in the concert room.

When the application is first launched, a set of « sprites » or icons appear in the background, rotating in the space. In the center of the screen, there is a central object, that moves along with the acceleration values, when orientating the device on the right/left sides or on the front/back. When the device is put in a flat surface, the central object is positioned in the center of the screen. At the bottom of the screen, there is a « Settings » button. When we click on it, a pop-up window appears. It shows the device identification (uuid), the current IP address and port used by the mobile device to send data to the Cubieboard.

There are also two inputs that enables the user to change these last values of IP address and port. The user can simply consulte the two parameters and close the window without restarting the application (« Close » button on the top right corner of the window). He/She can also decide to restart the application without modifying the parameters. In this case, he/she has to simply click on the button « Restart ». The pop-up window closes then and the application starts again. The last option is that the user can change one or both parameters (IP address or/and port) and restart the application the same way than before.

2 - How does it work? How to use it?

In this section, we will present you the requirements necessary for running the Manticore network and how to use Manticore.

2.1 Requirements

To run successfully Manticore, you will need the following item :

- MacOS with the Manticore project + Max/MSP (at least the version 5)
- Android smartphones with the mobile app
- Cubieboard devices
- Router (+switch) + ethernet cables

2.2 Notice - Instructions

2.2.1 Installation Manticore

On your MacOS, to install the Manticore project, a script has been implemented to install every packages necessary (Nodejs, Npm, Puredata, OSC protocol...).

Once you have successfully downloaded/picked up the project on your computer, navigate to the main folder called Manticore and double-click on the install_osx file. It should open a Terminal and installing all the requirements for the project.

Once the installation is done, a success message will be displayed on the terminal.

Plus, on Max/MSP software, you have to add the lib necessary for Manticore to your environment which can be found at :

```
# ~/Manticore/projects/Max/max_externals/lib/
```

If you need to update the whole project on your MacOS, just open the GitHub application and synchronise.

On each Cubieboard, the Manticore project will be installed and delivered with it. There is nothing to configure on it. When you will plug on the Cubieboard, Manticore will start automatically and start their own access point named Manticore_Device#X depending on which Cubieboard you are

On each Android smartphone, the mobile application will be installed and delivered with it. There is nothing to configure on it.

Nota: Wireless password for each Cubieboard is organised this way :
SSID: Manticore_Node#X
Password: azerty123

2.2.2 Running phase

1 - Plug on the router and connect all the Cubieboard to it using ethernet cable. Then, plug every other devices on (Cubieboard, Android smartphones, MacOS computers). For the Cubieboard, you have to nothing else since manticore project will start automatically.

2 - Start Manticore on MacOS: double click on the file start_manticore. It should start manticore network and open the web interface (http://localhost:3000) where you can monitor the whole network. You should have something like this:

Figure 2 - Web Interface of Manticore Network

Let's explain a bit what you have on the web interface:

1 - The name of the device where Manticore is running

2 - Number of nodes connected to the Manticore network

3 - Refresh button - It allows you to refresh the network (does one of the node or sensor is disconnected? Does the IP address has changed?). You can click on if you think there is something wrong in the network, for instance, if you have plug off one of the Cubieboard is still connected on the web interface.

4 - Node of the network with its features (ID, IP address)

5 - Sensor of the network with its features (ID, IP address)

6 - Sensors connected to the device running Manticore

7 - Records of requests made in the network

When you are clicking on the *refresh button (#3),* you have to wait around five seconds before refreshing the web page.

3 - Check if every android smartphones are connected to a wireless network named Manticore_Node#X. Otherwise, you have to manually connect the smartphone to wireless. and start the mobile application.

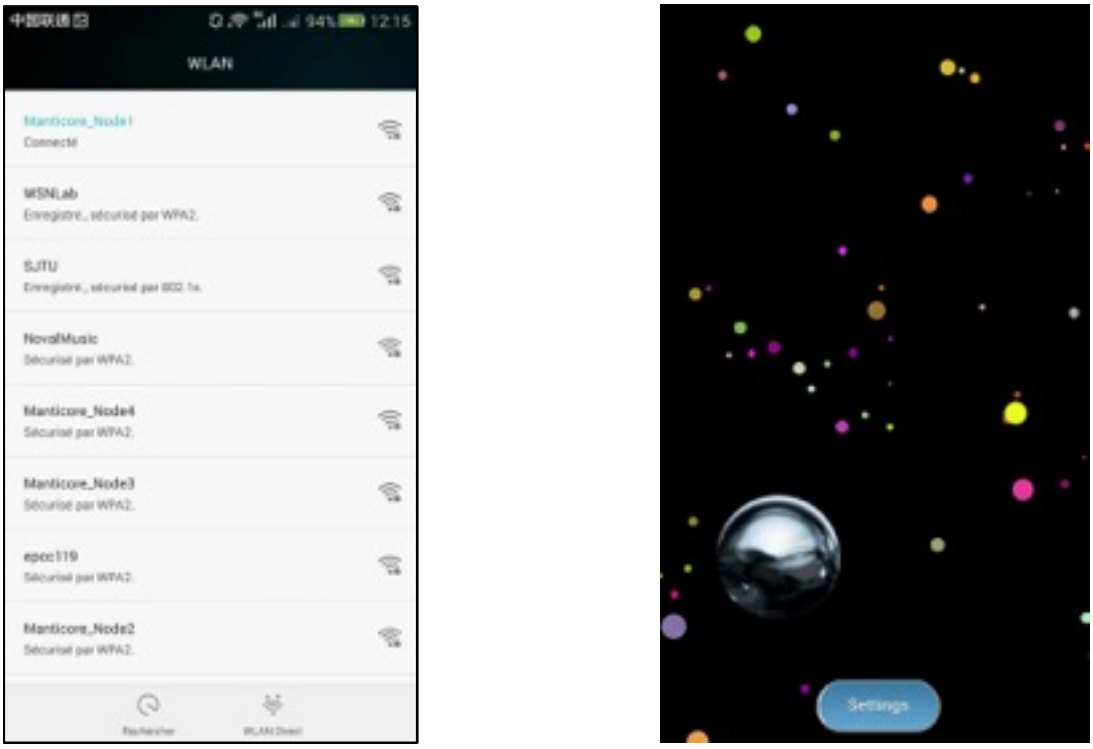


Figure 3 - Screenshot of the android smartphone & mobile application

4 - On the web interface (http://localhost:3000) you should observe the whole network, i.e., Cubieboard, MacOS, Smartphones.

5 - Start Max/MSP and run the right patcher that you want to use. In your patcher, you have to add an external called mxj NetScan in order to retrieve all the data from smartphones. This external NetScan.java can be found in the Manticore folder :

```
# ~/Manticore/projects/Max/max_externals/src/
```

6 - Specify the UDP port for each smartphone that you're using into Max/MSP. For example, if you want to retrieve the data from the smartphone #1, you will put: udpreceive object + port linked. An object is created automatically when you tick and ask for a data resource. If you want to release a resource previously requested, you have to untick the case.

/ ! \ When you want to request a ressource from Max/MSP, first, refresh the whole network using the refresh button on the web interface in order to prevent from any Cubieboard or smartphone disconnection. / ! \

Figure 4- Max/MSP interface with Manticore Network

8 - Once you have done with the network, you can shutdown every devices.

2.2.3 Uninstall Manticore

On your MacOS, to uninstall the Manticore project, a script has been implemented to uninstall every packages (Nodejs, Npm, Puredata, OSC protocol...). To do so, you have to navigate to the main folder and double-click uninstall_osx (this will open a terminal session).

Once the uninstallation is done, a success message will be displayed on the terminal.

On the Android smartphones, to uninstall the mobile application, you just have to delete it like if this is a traditional application.

3 - Configuration of the Cubieboard

In order to connect to a Cubieboard, you have to connect to it with SSH connexion. The Secure Shell, or SSH, is a cryptographic (encrypted) network protocol for initiating text-based shell sessions on remote machines in a secure way. This allows you to run commands on a machine's command prompt without them being physically present near the machine.

In order to find a Cubieboard on the network it's connected to, you can use the nmap software which allows you to detect any devices connected to the network :

```
YourMac'sName # nmap -p 22 Network_IP_@/24
```

For example you will have something like this if the network IP address is 192.168.100.0 :

```
YourMac'sName # nmap -p 22 192.168.100.0/24
```

This command will display all the devices and specify all the devices connected and the port 22 status (opened or closed).

So for example, if you want to connect to the first Cubieboard you should use the terminal and use the following command line:

```
YourMac'sName # ssh linaro@192.168.100.1 (192.168.100.1 is the IP address of the Cubieboard#1)
```

The terminal will ask for a password which is the default one: linaro.

Nota: the system could ask you if you want to save this SSH connexion into your Mac, just type "yes" and it will ask for the password.

Then, if you want to configure or modify: for any modification, we are using the vi software which is a default editor on Unix OS. Follow this link to know the basic command ("i" for insertion, "esc+ :wq" to save and quit, "esc+q!" to quit without saving).

- Access point: on our Cubieboard, we are using the *hostapd* software which allow us to create an AP. The configuration file can be found using the command line (first, you have to connect to it via SSH):

```
linaro@Cubieboard1 # sudo vi /etc/hostapd/hostapd.conf
```

- Network interfaces: you can change and configure both interfaces of the Cubieboard: eth0 which is the ethernet interface and the walnut which is the wireless interface (first, you have to connect to it via SSH).

```
linaro@Cubieboard1 # sudo vi /etc/network/interfaces
```

- Booting program: you can change which software you want to start during the booting of the cubieboard

```
linaro@Cubieboard1 # sudo vi /etc/rc.local
```

- Rebooting command: if you have any problem with the Cubieboard and if it still connected to the network, first, you have to connect to it via SSH and then, use this following command line to restart the device:

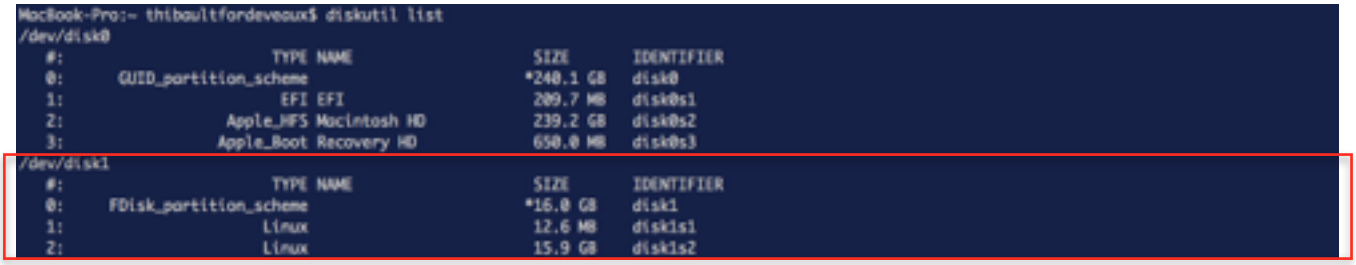
```
linaro@Cubieboard1 # sudo reboot
```

- Update the git repo: in order to update the Github folder (follow this link for any problem with the CLI). You have to be added to the private repo by Shilong first of all.

```
linaro@Cubieboard1 # cd $HOME/pfe
linaro@Cubieboard1 # git status  <— See modifications
linaro@Cubieboard1 # git pull   <— Update the folder
        > User_id
        > Password
```

- How to use the backup image: if you want to install a new image of the OS on a Cubieboard (you can follow this link)

```
YourMac'sName # diskutil list (or df -h)
```



This will list you all the storage devices plugged.

YourMac'sName # diskutil unmount /dev/disk1 (where disk1 is the SDCard)

YourMac'sName # sudo dd if=/path_to_backup_image/SDCardBackup_Manticore.img of=/dev/rdisk1 bm=1m (where rdisk1 is the SDCard)

This step will take a while like 5/10 minutes. After what you should have something like this displayed (numbers displayed are just an example):

2172752+0 records in

2172752+0 records out

1112449024 bytes transferred in 297.167711 secs (3743506 bytes/sec)

The last command line is necessary to eject the SDcard properly:

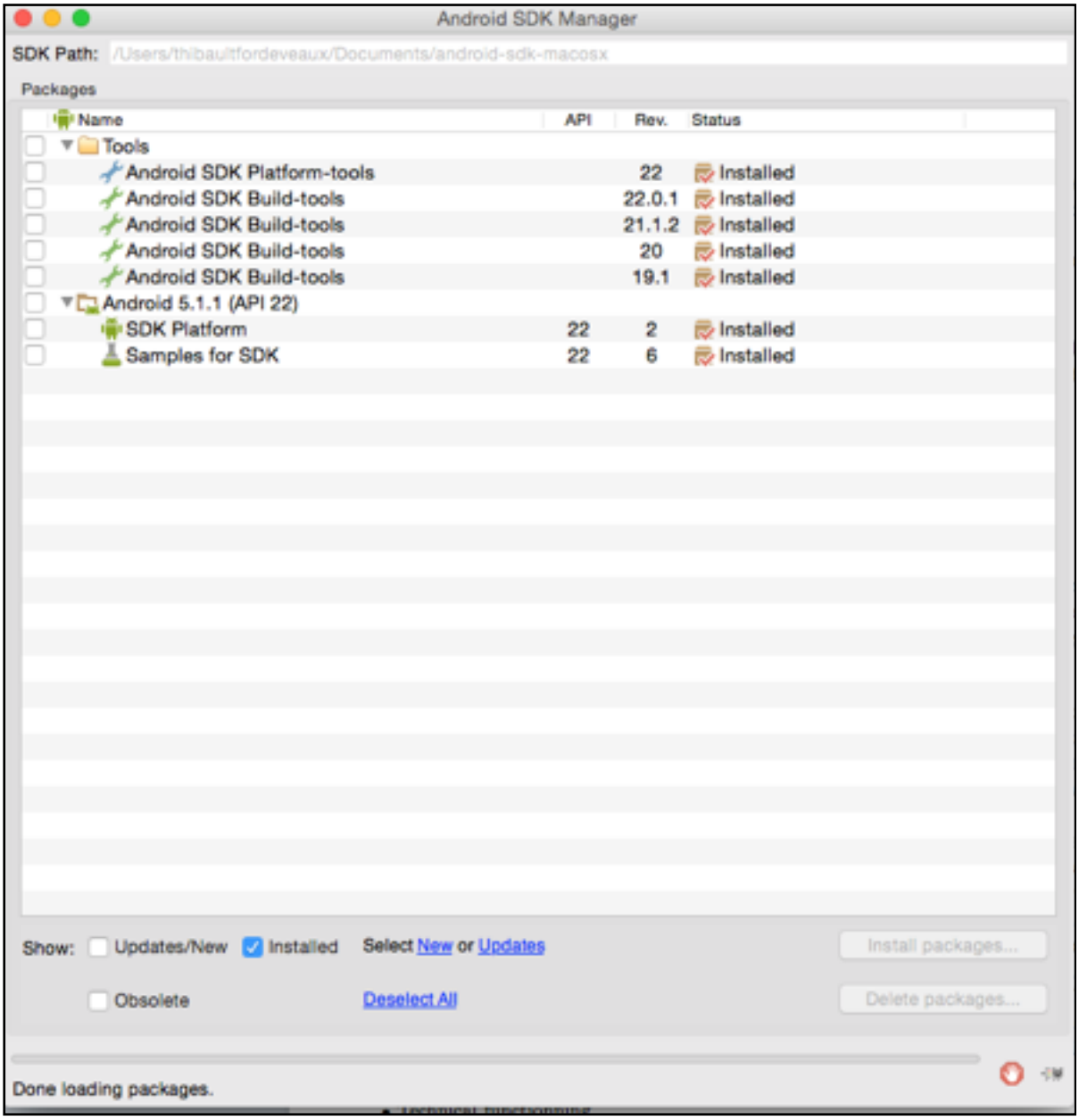
YourMac'sName # sudo diskutil eject /dev/rdisk3¶

4 - Configuration of the mobile application

The mobile application is using the Phonegap framework. All the command lines necessary in order to compile the project are in the previous link. Be aware of the code on GitHub repo is implemented for one computer, i.e., you have to change some files like *local.properties* (there are two files like this: one in android folder, the other one in android/CordovaLib/ folder) by putting the right path for the SDK used + the right path for the platform_tools and the tools for the SDK added into .bash file.

/ ! \ Before going further on the mobile application, you need to have install the SDK stand-alone first following this link. Then once you have the "Android-sdk-osx" folder, you have to install all the tools necessary using the SDK manager. Run this command into the "Android-sdk-osx" folder (this should display and open the Android SDK Manager):

```
tools/android update sdk
```



To develop the mobile application, I used a WebGL Framework called Three.js, that helps developers display 2D or 3D objects and animate them with special geometries, materials, effects and controls; on multiple Web platforms.
If needed, the reference Website of the Framework is : http://threejs.org, where you can find the well-explained documentation and play with demo examples. You can also find the source code of the examples following the link: https://github.com/mrdoob/three.js.

- How it works

When the application is launched, you can see a set of « sprites » or icons in the background, rotating in the space. In the center of the screen, there is also a central object, that moves along with the acceleration values, when orientating the device on the right/left sides or on the front/back. When the device is put in a flat surface, the central object is positioned in the center of the screen. At the bottom of the screen, there is a « Settings » button. When we click on it, a pop-up window appears. It shows the device identification (uuid), the current IP address and port used by the mobile device to send data to the Cubieboard. There are also two inputs that enables the user to change these last values of IP address and port. The user can simply consulte the two parameters and close the window without restarting the application (« Close » button on the top right corner of the window). He/She can also decide to restart the application without modifying the parameters. In this case, he/she has to simply click on the button « Restart ». The pop-up window closes then and the application starts again. The last option is that the user can change one or both parameters (IP address or/and port) and restart the application the same way than before.

- Technical functionning

All the application code has been implemented in the file « index.js ».
Note: in the « js » directory, there is another file called « index2.js ». It refers to the previous version of the user interface (with colorful particles spread out in the background and a central ball moving across the screen). Normally, this version is not needed anymore.

function init()

This function is used to initiate the user interface: define all its component Three.js (camera, two scenes, two meshs, two CSS2DObjects, two renderers) and add event listeners to be called when the window is resized (from portrait mode to landscape mode) and/or the « Settings » and « Restart » buttons are clicked. It also calls the « addSpritesInGroup » and « createRenderers » functions.

function addSpritesInGroup(group)

This function is the one that creates the background objects called « Sprite ». To add the moving particles in the background, we are using a set of images (that you can find in the directory img/ highway-traffic-icons). The path to every image is defined in the global variable « imageSrcs », that is an array. In this function, we loop through the array and create a Sprite for every image contained in the directory.

function animate()

Once we have called the « init » function to set up the scenes and objects that we want to display, we call the « animate » method to render our scenes 60 times per second. In turn, it calls the « render » method that allows the camera to point out on the main scene and move the background objects by making rotations around the X and Y axis (if you want to move the background objects in a different way, here is where you should do it).

function getAcceleration()

It starts by getting the current IP address and current port defined in the global variables with the same names (still at the beginning of the « index.js » file); and use them to constitute the address used to send accelerometer data to (192.168.42.1 and 8081 by default). A WebSocket is created, then opened. We then watch the accelerometer values. Every time these values change, the success callback is called. In this callback, the « updateAnimation » function is called and the OSC message is sent with the accelerometer data, via the WebSocket.

function updateAnimation()

This function is used to update the position of the central object and make it move across the screen along with the accelerometer values changes. To make it move, we translate it each time using a 3D Vector. Three conditions have been set up:

- if the accelerations in the X axis AND the Y axis are approximately equal to zero (use of « Math.floor » to get the integer values), the central object is put back to its initial position, which is the center of the screen. (cssObject.position.x = planeMesh.position.x; cssObject.position.y = planeMesh.position.y;)
- 2 conditions have also been defined to make sure the central object never cross the screen limits (total width and 2/3 height of the screen - note: the last 1/3 height of the screen is used to display the « Settings » button and we don't want the central object to intersect with the button during its trajectory).

Also, the two screen modes are considered: portrait and landscape; although nothing has been implemented for the landscape mode (only put there if needed).

function restartApplication()

This function is called when the « Restart » button is clicked in the pop-up Settings window. The explanation is detailed with comments in the source code.

All the other functions that come at the end of the « index.js » file are used to generate the pop-up and give it the right size (considering the screen size).

• Possible changes - How to ... ?

Change the icons of the background particles

Add your image in the directory « img/highway-traffic-icons » and add the path to your image in the array « imageSrcs » defined as a global variable in « index.js ».

Change the image of the central object

Add your image in the « img » directory. Add the path to this image in the canvas « img » (with id=« show ») in the file « index.html ».

Warning: if the dimensions of the image are too big, it's possible that it will not appear as

expected on the mobile screen. The dimensions of the image have to be approximately equal to 128*128, so if your image is too big, go to the « index.css » file and search for the definition of the object with id « show » (#show { ... }), replace the existing percentages of width and height to fit the previous expectations. For example, for the central object, I used a 2400*2400 image. To make it 128*128, I defined, in the index.css file: #show { width: 18%; height:18%; }, which means: use only 18% of the image width and height.

- Possible improvements

The artists asked me to create an effect on the screen when the mobile device is shacked. The task is not easy, but you can refer to « Blur » or « Smoke » effects in Three.js.
Here are also some links that could be handy, in case this functionality needs to be added:

- http://stackoverflow.com/questions/15354117/three-js-blur-the-frame-buffer
- http://ilee.co.uk/shake-gesture-detection-in-phonegap/
- http://code.tutsplus.com/tutorials/webgl-with-threejs-textures-particles--net-35836

- Possible problems - Quick loss of device battery

If a quick loss of battery is noted in the mobile devices, two possible explanations have to be considered:
- a too big amount of particles in the background can affect the battery of the mobile device (for that, the number of images (or sources to images) in the global variable « imageSrcs » has to be decreased.)

- to create the animations, we need what's called a render loop. The function « animate » creates a loop that causes our renderers (renderer and cssRenderer) to draw two scenes 60 times per second. We could use « setInterval » instead of requestAnimationFrame, but requestAnimationFrame has a number of advantages. Perhaps the most important one is that it pauses when the user navigates to another browser tab, hence not wasting their precious processing power and battery life.

To sum-up : if you see that the mobile devices are loosing too quickly their batteries, try to use
« setInterval » instead of « requestAnimationFrame », or try to reduce the number of particles in the background.

To optimize the performances, you can also refer to the following link:
Optimizing Three.js Performance: Simulating Tens Of Thousands Of Independent Moving

Objects: http://www.ianww.com/blog/2012/11/04/optimizing-three-dot-js-performance-simulating-

tens-of-thousands-of-independent-moving-objects/