Appendix D

Mobile Applications Configurations

D.1 UAV Teleoperation application

My scenario is based on the Apple's iOS device since I have a previous experience on coding for this ecosystem. The prototype application is visually structured for the layout of iPhone 5 (4 inches diagonal, 1136x440, 326ppi). The targeted version iOS version is 8.

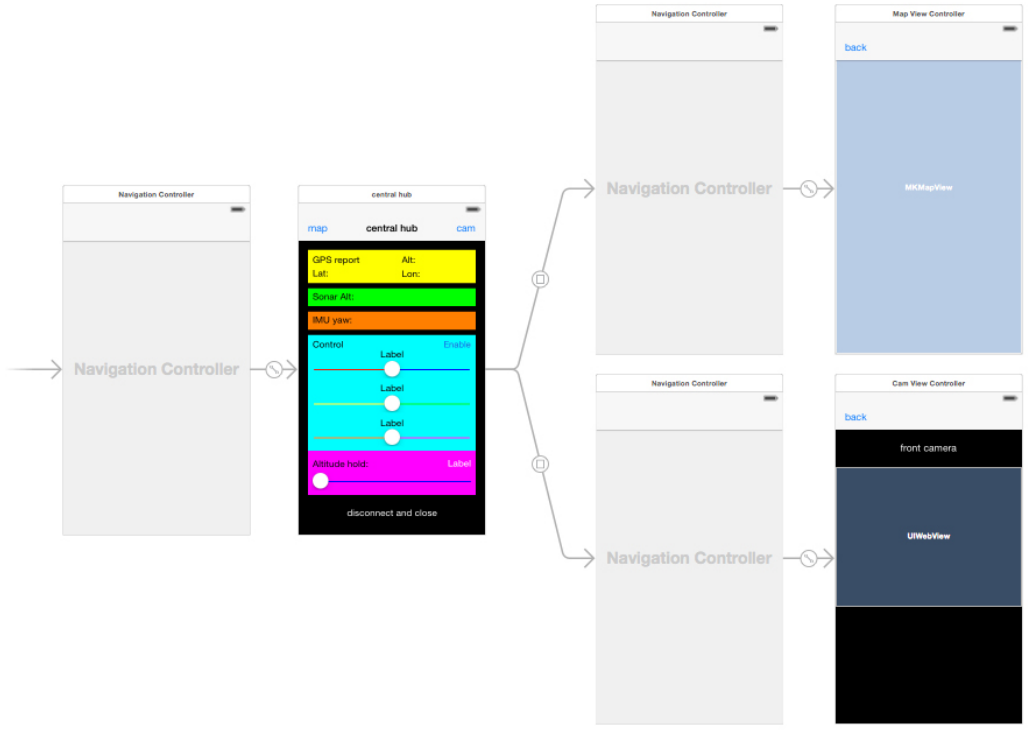

Fig.D.1 The storyboard of the teleoperation application

The implementation of the application was based on three *UIViewControllers* embedded in navigation controllers (fig.D.1). The navigation between the

screens was accomplished using segues. In the projects' workspace, three external libraries have been imported, namely:

- SocketRocket
- RBManager
- MKMapView

As long as the control tab is enabled, the control messages are published in a continuous rate of 10Hz using an *NSTimer* object.

On the other side ROS was executed in a VMWare Virtual Machine. The ROS distribution was Indigo and the operating system Ubuntu 14.04 LTS. The simulation was based on Hector quad-rotor ROS stack. The host computer was connected with the iPhone via a Wi-Fi access point. The network topology is shown in figure (fig.D.2).

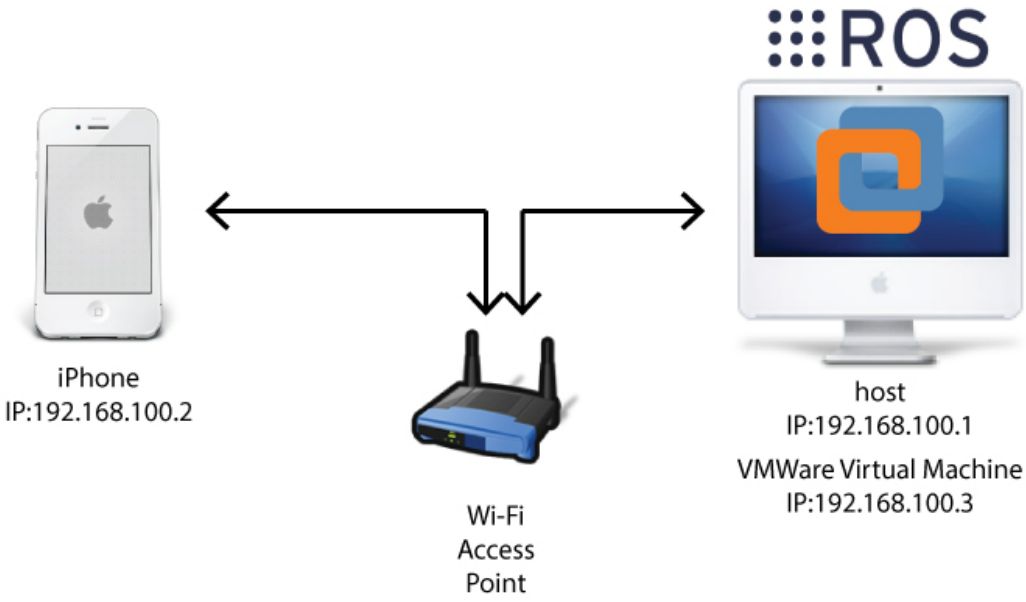

Fig.D.2 The network topology of the development environment

For a successful working scenario the *my_hector* package, which was developed for the assisted obstacle avoidance flight mode, had to be installed. Particularly, the hover node had to be executed to allow the altitude hold mode (*rosrun my_hector hover*). In addition, the *Rosbridge* and the *MJPEG*

server also had to be turned on and to be running using the following commands:

- *roslaunch rosbridge_server rosbridge_websocket.launch*
- *rosrun mjpeg_server mjpeg_server*

The iOS source files of the project are available online in the project's github repository (goo.gl/TKlkE5)

D.2 Radiation mapping application

Similarly to the previous one, the radiation mapping application was developed based on the iPhone 5 screen dimensions. Consequently, the layout and the size of the elements are adapted to this device. The minimum supported iOS version is the 7th.

The implemented application consists of three *UIViewControllers* embedded in navigation controllers (fig.D.3). The main data structure that holds the mapping data is an *NSArray.* The elements, which the mapping array stores, are objects that belong to the custom class *Measurement.* The advantages of using an array are: the potential future integration of a save and load mechanism as well as the easy handling of data as a set in the map visualization (redraw/clear).

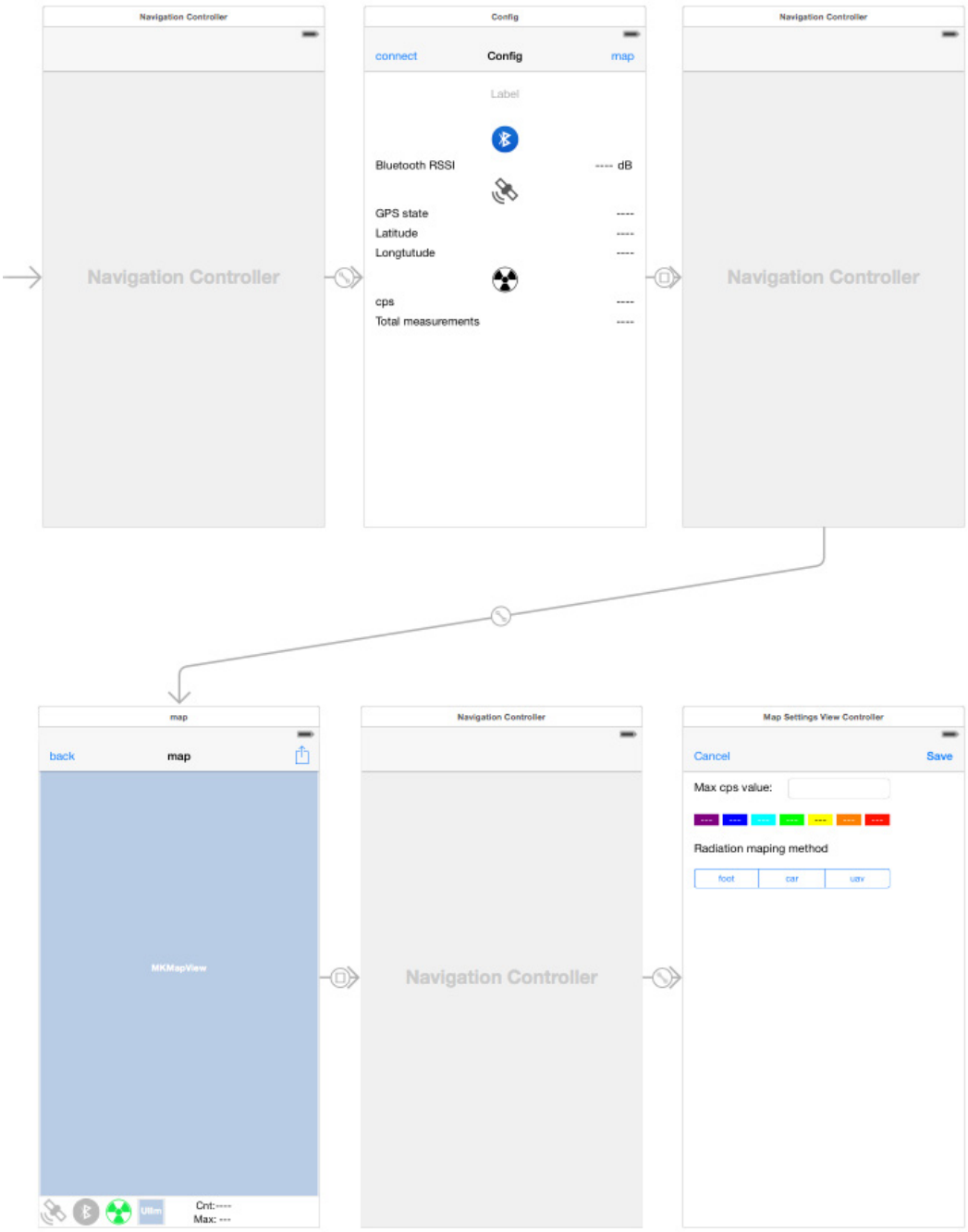

Fig.D.3 The storyboard of the radiation mapping application

The first screen is called "Configuration". The related *UIViewController* is mainly responsible to handle the Bluetooth connection. It includes the entire set of the related methods, which are provided by the BLE library. When the user clicks the "Connect" button, using the appropriate sequence of methods' execution, the Bluetooth connection is activated if a compatible device is enabled within the range. Afterwards, having established the connection, the

application receives the messages, which are transmitted from the measurement unit. The *UIViewController* has all the required tools to parse the payload of the messages. More precisely, at the beginning, two data structures are defined in order to handle binary floating-point numbers (32bit) and binary integer numbers (16bit). When the specific bit of the status byte gets enabled, the parsing of the raw bytes begins. The payload of the message contains information about the number of connected satellites, the longitude, the latitude and the CPS value. The last three elements are stored in a newly created *Measurement* object and then, in turn, this object is stored in the mapping array.

The next screen is the map. It can be switched on, on that screen, only if the GPS is locked. One noteworthy mechanism that has been implemented is the share of the same BLE manager with the previous *UIViewController*. Particularly at the transition, the same BLE manager is passed to the map screen. This is desirable because map screen needs to receive messages and status information for the summary section. Before the transition is made, the reception of new messages is disabled in the previous screen and enabled in the new screen and vise versa for the reverse switch. In case of having individual BLE managers means that in the new screen the Bluetooth connection would have to be established again, which is unacceptable. Not only do the two screens share the BLE managers but they also share the mapping array. Consequently, this array is unique in the application and is passed to each active *UIViewController.*

Another important characteristic in the implementation of the map screen is the approach that enables the individual color of each point. The standard method of using *MKCircle* overlays does not support the selection of color for each point. It is possible to select only the color for the entire overlay. To extend the provided class, I used class inheritance and I sub-classed the *MKCircle* to create *MyCircle*. This approach allowed me to use the same standard visualization mechanism and at the same time satisfy the enhanced requirements.

The last noteworthy task implemented in this screen was the adaptive zoom level on the map. According to the value that the user selects in the next screen, the zoom level and the size of the circle, which represent the measurement, change. To reflect the change, all the measurements that are contained in the mapping array need to be redrawn. This is feasible using the developed helper methods.

In the third screen the user is provided with the option to select the maximum range of the CPS values color range and the mapping methods. The functionality is simple and when the user clicks the save button, the selected values return to the map screen and update the related features accordingly.

All in all, the navigation between the *UIViewControllers* is implemented using segues. The exchange of the data between them is accomplished using delegate methods.

On the other side the measurement unit consists of an Arduino Mega ADK as the base microcontroller and three shields (fig.D.4). The installed shields are namely:

- RedBear's lab BLE Shield
- Wireless SD Shield
- Adafruit Ultimate GPS

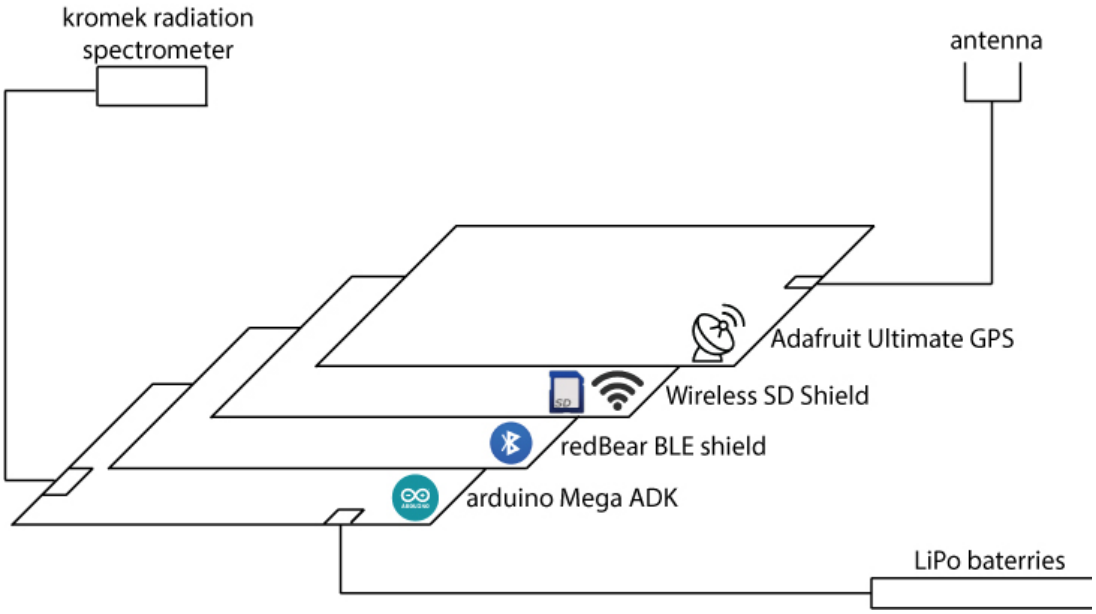

Fig.D.4 The hardware profile of the measurement unit

The Arduino sketch includes many external libraries in order to successfully support the operation of all the individual interfaces. The required libraries are:

- Adafruit GPS Library
- BLE
- RBL nRF8001
- TinyGPS
- USB Host Shield v2.0

One of the advanced features implemented in this Sketch is the break of the native data types to bytes in order to compose the payload of the message. The approach is very similar to what is implemented in the iOS side. Likewise, two data structures are defined in the beginning of the script. The first supports binary 32bits floats and the second binary 16bit integers. The data, using this mechanism, are broken to bytes and sent sequentially by employing the BLE shield.

It is important to cite that the initial IAC sketch was compatible with the Arduino 1.0.x IDE and not with the most recent 1.5.x and 1.6.x. versions. In order to avoid the required time to build the sketch from scratch, since it was

not working with the newer versions, I adapted the 1.0.x version by downgrading the BLE and the RBL nRF8001 libraries. This way I maintained the compatibility, in order to have a working set of code.

The iOS and Arduino source files of the project are available online in the project's github repository (goo.gl/565h4M).