InPresS Mobile

Acquisition, processing and display of analog sensor signals developed in Android and Arduino.

Víctor Andía Sierra.

Supervisor
Ph.D Marco Balsi

Project Owner
Dr. Franca Tecchio

September 2010
I would like to thank my supervisor Marco Balsi and his assistant Salvatore Esposito for providing the motivation and resources for this project to be done. Without them this thesis would not have been possible. Besides, I want to show my appreciation to the group LET’S from Ospedale Fatebenefratelli di Roma led by Dr. Franca Tecchio for the great work done. Also, I would like to mention Marco Rossi and Miro Barsocchi from Movenda and Antonio Castellon from Ebotlution Systems for their unconditional support.
# Table of contents

1. **Introduction**  
   1.1 Motivation  
   1.2 Goal  
   1.3 Overview  

2. **InPresS Mobile, an Android app**  
   2.1 What is InPresS?  
   2.2 Using InPresS  

3. **Android Development**  
   3.1 A brief history of Android  
   3.2 Required tools  
   3.3 Application architecture  
   3.4 Activity  
   3.5 Intent and Intent Filters  
   3.6 Intent Receiver  
   3.7 Service  
   3.8 Content Provider  
   3.9 Android User Interfaces  
   3.10 Hierachy of Screen Elements
3.11 Views 16
3.12 Viewgroups 16
3.13 A Tree-structured UI 17
3.14 Comparing Android UI Elements to Swing UI Elements 18
3.15 The AndroidManifest.xml 18
3.16 General 19

4 Arduino Development 23

4.1 Hardware 23
4.2 Software 24
4.3 Arduino BT 25
4.4 Summary 26
4.5 Schematic 26
4.6 Power 27
4.7 Input and Output 27
4.8 Bluetooth Communication 28
4.9 Communication 28
4.10 Information 29
4.11 Using the Arduino BT 29

5 Amarino, “Android meets Arduino” 30

5.1 Documentation 30
### 5.2 Download

5.3 Connecting Android to Arduino

5.4 Android setup

5.5 Arduino setup

5.6 Monitoring

5.7 Real-time data

#### 6 Conclusions

#### A Source Code

| A.1 | AndroidManifest.xml | 40 |
| A.2 | /inPress/src/com.PFC.inpress | 41 |
| A.2.1 | about.java | 41 |
| A.2.2 | info.java | 41 |
| A.2.3 | LevelView.java | 42 |
| A.2.4 | LevelView2.java | 43 |
| A.2.5 | main.java | 44 |
| A.2.6 | screen1.java | 45 |
| A.2.7 | screen2.java | 47 |
| A.3 | /inPress/res/layout | 59 |
| A.3.1 | about.xml | 59 |
| A.3.2 | main.xml | 60 |
| A.3.3 | screen1.xml | 62 |
A.3.4    screen2.xml  62
A.4    /inPress/res/values  64
    A.4.1    strings.xml  64
A.5    /inPress/res/drawable  65
A.5    Arduino code  65

Bibliography  66
# Table of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>WorldWide Smartphone Sales in 2010</td>
<td>1</td>
</tr>
<tr>
<td>2.1</td>
<td>Complete structure of the system</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>InPresS Beta Version</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>Android main menú</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>InPresS Mobile main menú</td>
<td>6</td>
</tr>
<tr>
<td>2.5</td>
<td>ABOUT US screen</td>
<td>6</td>
</tr>
<tr>
<td>2.6</td>
<td>InPresS Mobile looking for GPS position</td>
<td>7</td>
</tr>
<tr>
<td>2.7</td>
<td>Telnet sending the GPS coordinate to the Android emulator</td>
<td>7</td>
</tr>
<tr>
<td>2.8</td>
<td>InPresS find GPS position</td>
<td>8</td>
</tr>
<tr>
<td>2.9</td>
<td>InPresS configured correctly</td>
<td>8</td>
</tr>
<tr>
<td>2.10</td>
<td>InPresS configuration parameters</td>
<td>9</td>
</tr>
<tr>
<td>2.11</td>
<td>InPresS being used in medical practice</td>
<td>9</td>
</tr>
<tr>
<td>2.12</td>
<td>InPresS sending data by mail to the hospital</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Android Activity life cycle</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>Android Views structure</td>
<td>17</td>
</tr>
<tr>
<td>4.1</td>
<td>Arduino BT board</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Arduino BT schematic</td>
<td>26</td>
</tr>
<tr>
<td>5.1</td>
<td>Amarino logo</td>
<td>30</td>
</tr>
<tr>
<td>5.2</td>
<td>Amarino’s first screen</td>
<td>32</td>
</tr>
<tr>
<td>5.3</td>
<td>Bluetooth connections</td>
<td>33</td>
</tr>
<tr>
<td>5.4</td>
<td>Adding a new bluetooth connection</td>
<td>33</td>
</tr>
<tr>
<td>5.5</td>
<td>Pairing request</td>
<td>34</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.6</td>
<td>Introducing bluetooth PIN</td>
<td>34</td>
</tr>
<tr>
<td>5.7</td>
<td>Amarino connected</td>
<td>35</td>
</tr>
<tr>
<td>5.8</td>
<td>Adding events</td>
<td>35</td>
</tr>
<tr>
<td>5.9</td>
<td>Adding events</td>
<td>35</td>
</tr>
<tr>
<td>5.10</td>
<td>Adding events</td>
<td>36</td>
</tr>
<tr>
<td>5.11</td>
<td>Adding events</td>
<td>36</td>
</tr>
<tr>
<td>5.12</td>
<td>Arduino development environment</td>
<td>36</td>
</tr>
<tr>
<td>5.13</td>
<td>Monitoring Arduino processes in Arduino</td>
<td>38</td>
</tr>
<tr>
<td>A.1</td>
<td>Eclipse development environment</td>
<td>40</td>
</tr>
<tr>
<td>A.2</td>
<td>Images used in InPresS Mobile</td>
<td>65</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

In recent years, due to the rise of new technologies, smart phone users have increased much more than any technology sector.

At first, with the advent of iPhone, users began to think and believe that the use of a smart phone with 3G connectivity could help improve their lives. Today, these thoughts are becoming a reality and new technologies are present in every corner of the planet.

On October 21, 2008 was born Android as a variant of Linux-oriented mobile devices. Today is already a reality in mind and already has more users than iPhone and Windows Mobile.

Moreover, forecasts indicate that the growth of Android over other competitors will be much higher in coming years.

The reason why Android to get more customers than its competitors in such a short time is due to the policy of the company that created it, Google.

Android, like most Google products, is open source, this means that any developer can access the source code so you can develop own applications or improve the operating system.
Another major advantage of Android is that being an open source operating system can be installed on any mobile device so you can enjoy the desired characteristics such as GPS, a camera, sensors ...
All these features make it a great Android operating system to develop technological applications because they can enjoy the powerful capabilities at very affordable prices.

### 1.2 Goal

The aim of this project is to implement InPresS in the Android platform.
InPresS is a medical application developed by the team LET’s from Ospedale Fatebenefratelli di Roma, which takes data from an analog sensor that the patient must press. The application in addition to acquiring these data, processes and displays them on screen.
In this project, we design an application, that in addition to the actions that had inpress beta version, it has other very useful options such as GPS references and full connectivity between patient and hospital.
The main goal we set ourselves in this project is to improve the lives of patients who must use the device for safety and comfort.

### 1.3 Overview

This report is organized as follows: first, we have explained that it is Inpress in essence, what are its characteristics and how to use, in Chapter 2. Then we explained how to developeon the platforms we used, in Chapter 3 how to develop Android, in Chapter 4 how to develop on Arduino and Chapter 5 how to use the library Amarino. Then we include the source code and finally the conclusions.
Chapter 2

InPresS Mobile, an Android app.

2.1 What is InPresS?

InPresS is a portable device that produces, acquires and displays the pressure exerted by the user through a sensor. It is developed under the Android and Arduino platforms.

The data are acquired by a pressure sensor connected to an Arduino board that communicates via bluetooth with the Android device.

InPresS is a particular application but the system is designed to be used with all types of analog sensors which need treatment and visualization of their data.

In fact, the Android device is responsible for processing and displaying data and the Arduino board is only responsible for taking analog sensor data and send them by bluetooth to the phone.
The initial solution in which the working group thought the sensor was directly connected to the phone via Bluetooth but this is not possible in current versions of Android because until version 2.2 which has not yet been released Bluetooth is only operating be connected to handsfree devices. That is why impress used in its development Amarino library which allows you to connect via Bluetooth Android and Arduino.

InPresS Android version is preceded by a beta version developed in LabView, this work was done by the group LET'S from Ospedale Fatebenefratelli di Roma.

![InPresS Beta Version.](image)

The basic requirements to use InPresS are:

- Android phone. Android OS minimum version required is Android 2.1. (we have used HTC TATTOO with the unofficial sdk version)
- Arduino board with Bluetooth connectivity. (we have used Arduino BT with Bluetooth antenna integrated)
- Pressure sensor. (we have used Honeywell 40PC015G1A)

As discussed before, InPresS is a device that acquires data from an analog sensor and processes. The algorithm that governs the device is as follows:

First of all, the devices has to be configurated, this configuration is based on data collection and the subsequent calculation of the maximum and minimum of three different intervals. That maximum is important because is the Maximal Voluntary Contraction (MVC).

When you have the values is time to calculate the 5% of the average value of these maximums. Obviously you have to take into account the minimum value because it is the value that refers to the off position and
must be subtracted from the values with which we work. This 5% is the threshold at which the user should try to fix the color bar in the medical practice.

Inpress is also able to locate the patient by GPS and send all data taken by the application (settings and exercise) by email to the hospital. In this way, the hospital always know the geographical position of the patient when doing the exercise and they can check through the records received his medical condition and they can take appropriate action.

### 2.2 Using InPresS.

To use InPresS, first of all, press the HOME button to access the application menu.

![Android main menu](image)

**Figure 2.3 - Android main menu.**

Then, you should press the FBF_InPresS icon which allows you to use the app.

The first step when you are using InPresS is write down your name. That name is the identification that the hospital will receive with all your information.
On the first screen you can see the ABOUT US button, this button has the main information of the developer and the patient’s hospital.
On the other hand, if you press the START button another screen appears. That screen looks for the GPS position.

![Figure 2.6 - InPresS Mobile looking for GPS position.](image)

If you are running FBF_InPresS on the Android emulator because you are a developer you should send the GPS position by telnet.

![Figure 2.7 - Telnet sending the GPS coordinate to the Android emulator.](image)
A few seconds later the GPS position appears.

![Figure 2.8 - InPresS find GPS position.](image1)

The next step is to setup the device. The setup process is very easy, the patient must press and release the sensor firmly until the end of the configuration process. Then, the program will show the values calculated.

![Figure 2.9 - InPresS configured correctly.](image2)
Then, the program will show the values calculated.

Figure 2.10 - InPresS configuration parameters.

Right now the device is configured and is ready to use it. To use inPresS the patient will try to get the red color on the graph.

Figure 2.11 - InPresS being used in medical practice
When the patient ends the exercise should press the BACK button to send to the hospital the files (DATA FILE, CONFIGURATION FILE) by mail.

Figure 2.12 - InPressS sending data by mail to the hospital.
Chapter 3

Android Development.

3.1 A brief history of Android

The Android platform is the product of the Open Handset Alliance, a group of organizations collaborating to build a better mobile phone. The group, led by Google, includes mobile operators, device handset manufacturers, component manufacturers, software solution and platform providers, and marketing companies. From a software development standpoint, Android sits smack in the middle of the open source world.

The first Android-capable handset on the market was the G1 device manufactured by HTC and provisioned on T-Mobile. The device became available after almost a year of speculation, where the only software development tools available were some incrementally improving SDK releases. As the G1 release date neared, the Android team released SDK V1.0 and applications began surfacing for the new platform. To spur innovation, Google sponsored two rounds of "Android Developer Challenges," where millions of dollars were given to top contest submissions. A few months after the G1, the Android Market was released, allowing users to browse and download applications directly to their phones. Over about 18 months, a new mobile platform entered the public arena.

3.2 Required tools

The easiest way to start developing Android applications is to download the Android SDK and the Eclipse IDE. Android development can take place on Microsoft® Windows®, Mac OS X, or Linux. This article assumes you are using the Eclipse IDE and the Android Developer Tools plug-in for Eclipse. Android applications are written in the Java language, but compiled and executed in the Dalvik VM (a non-Java virtual machine). Coding in the Java language within Eclipse is very intuitive; Eclipse provides a rich Java environment, including context-sensitive help and code suggestion hints. Once your Java code is compiled cleanly, the Android Developer Tools make sure the application is packaged properly, including the AndroidManifest.xml file.

It's possible to develop Android applications without Eclipse and the Android Developer Tools plug-in, but you would need to know your way around the Android SDK.
The Android SDK is distributed as a ZIP file that unpacks to a directory on your hard drive. Since there have been several SDK updates, it is recommended that you keep your development environment well organized so you can easily switch between SDK installations. The SDK includes:

- android.jar
Java archive file containing all of the Android SDK classes necessary to build your application.

- documentation.html and docs directory
The SDK documentation is provided locally and on the Web. It's largely in the form of JavaDocs, making it easy to navigate the many packages in the SDK. The documentation also includes a high-level Development Guide and links to the broader Android community.

- Samples directory
The samples subdirectory contains full source code for a variety of applications, including ApiDemo, which exercises many APIs. The sample application is a great place to explore when starting Android application development.

- Tools directory
Contains all of the command-line tools to build Android applications. The most commonly employed and useful tool is the `adb` utility (Android Debug Bridge).

- usb_driver
Directory containing the necessary drivers to connect the development environment to an Android-enabled device, such as the G1 or the Android Dev 1 unlocked development phone. These files are only required for developers using the Windows platform.

Android applications may be run on a real device or on the Android Emulator, which ships with the Android SDK.

### 3.3 Application architecture.

In this chapter we will concentrate on the specialties in creating any Android-Application.

There are four building blocks to an Android application:

- Activity
- Intent Receiver
- Service
- Content Provider

Not every application needs to have all four, but your application will be written with some combination of these.
Once you have decided what components you need for your application, you should list them in a file called `AndroidManifest.xml`. This is an XML file where you declare the components of your application and what their capabilities and requirements are. We will discuss soon, what the `AndroidManifest.xml` is responsible for.

### 3.4 Activity.

Activities are the most common of the four Android building blocks. An activity is usually a single screen in your application. Each activity is implemented as a single class that extends the `Activity` base class. Your class will display a user interface composed of Views and respond to events. Most applications consist of multiple screens. For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be implemented as an activity.

Moving to another screen is accomplished by a starting a new activity. In some cases an Activity may return a value to the previous activity - for example an activity that lets the user pick a photo would return the chosen photo to the caller.

When a new screen opens, the previous screen is paused and put onto a history stack. The user can navigate backward through previously opened screens in the history. Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain. Android retains history stacks for each application launched from the home screen.

The following diagram illustrates these loops and the paths an activity may take between states. The colored ovals are major states the activity can be in. The square rectangles represent the callback methods you can implement to perform operations when the activity transitions between states.

![Android Activity life cycle](image-url)
3.5 Intent and Intent Filters.

Android uses a special class called Intent to move from screen to screen. Intent describe what an application wants done. The two most important parts of the intent data structure are the action and the data to act upon. Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI). For example, to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website-URI.

```java
new Intent(android.content.Intent.VIEW_ACTION, ContentURI.create("http://anddev.org"));
```

There is a related class called an IntentFilter. While an intent is effectively a request to do something, an intent filter is a description of what intents an activity (or intent receiver, see below) is capable of handling. An activity that is able to display contact information for a person would publish an IntentFilter that said that it knows how to handle the action VIEW when applied to data representing a person. Activities publish their IntentFilters in the AndroidManifest.xml file.

Navigating from screen to screen is accomplished by resolving intents. To navigate forward, an activity calls startActivity(myIntent). The system then looks at the intent filters for all installed applications and picks the activity whose intent filters best matches myIntent. The new activity is informed of the intent, which causes it to be launched. The process of resolving intents happens at run time when startActivity is called, which offers two key benefits:

- Activities can reuse functionality from other components simply by making a request in the form of an Intent.
- Activities can be replaced at any time by a new Activity with an equivalent IntentFilter.

3.6 Intent Receiver

You can use an IntentReceiver when you want code in your application to execute in reaction to an external event, for example, when the phone rings, or when the data network is available, or when it's midnight. Intent receivers do not display a UI, although they may display Notifications to alert the user if something interesting has happened. Intent receivers are also registered in AndroidManifest.xml, but you can also register them from code using Context.registerReceiver(). Your application does not have to be running for its intent receivers to be called; the system will start your application, if necessary, when an intent receiver is triggered. Applications can also send their own intent broadcasts to others with Context.broadcastIntent().
3.7 Service

A Service is code that is long-lived and runs without a UI. A good example of this is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, the media player activity could start a service using `Context.startService()` to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. (You can learn more about the priority given to services in the system by reading Life Cycle of an Android Application.) Note that you can connect to a service (and start it if it's not already running) with the `Context.bindService()` method. When connected to a service, you can communicate with it through an interface exposed by the service. For the music service, this might allow you to pause, rewind, etc.

3.8 Content Provider

Applications can store their data in files, a SQLite database, preferences or any other mechanism that makes sense. A content provider, however, is useful if you want your application's data to be shared with other applications. A content provider is a class that implements a standard set of methods to let other applications store and retrieve the type of data that is handled by that content provider.

3.9 Android User Interfaces

User Interfaces (UI) in Android can be built within two ways, by defining XML-Code or by writing Java-Code. Defining the GUI structure in XML is highly preferable, because as one knows from the Model-Viewer-Control principle that the UI should always be separated from the program-logic. Additionally adapting a program from one screen-resolution to another is a lot easier. Defining a UI in XML is very similar to creating a common HTML- document, where you have i.e. such a simple file:

```html
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    The content of the body element.
  </body>
</html>
```
Just the same as in Android’s XML-Layouts. Everything is well structured and can be expressed by tree-structures:

```xml
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World"/>
</LinearLayout>
```

### 3.10 Hierarchy of Screen Elements.

The basic functional unit of an Android application is the activity-an object of the class `android.app.Activity`. An activity can do many things, but by itself it does not have a presence on the screen. To give your activity a screen presence and design its UI, you work with views and viewgroups - basic units of user interface expression on the Android platform.

### 3.11 Views.

A view is an object extending the base class `android.view.View`. It’s a data structure whose properties store the layout and content for a specific rectangular area of the screen. A View object handles measuring, its layout, drawing, focus changes, scrolling, and key/gestures for the screen area it represents. The View class serves as a base class for all widgets - a set of fully implemented subclasses that draw interactive screen elements. Widgets handle their own measuring and drawing, so you can use them to build your UI more quickly. The list of widgets available includes i.e. TextView, EditText, Button, RadioButton, Checkbox, ScrollView, ...

### 3.12 Viewgroups.

A viewgroup is an object of class `android.view.Viewgroup`. As its name indicates, a viewgroup is a special type of view object whose function is to contain and manage a subordinate set of views and other
viewgroups. Viewgroups let you add structure to your UI and build up complex screen elements that can be addressed as a single entity.

The ViewGroup class serves as a base class for layouts - a set of fully implemented subclasses that provide common types of screen layout. The layouts give you a way to build a structure for a set of views.

### 3.13 A Tree-Structured UI.

On the Android platform, you define an Activity's UI using a tree of view and ViewGroup nodes, as shown in the diagram below. The tree can be as simple or complex as you need to make it, and you can build it up using Android's set of predefined widgets and layouts or custom view types that you create yourself.

![Android Views structure](image)

To attach the tree to the screen for rendering, your Activity calls its `setContentView()` method and passes a reference to the root node object. Once the Android system has the reference to the root node object, it can work directly with the node to invalidate, measure, and draw the tree. When your Activity becomes active and receives focus, the system notifies your activity and requests the root node to measure and draw the tree. The root node then requests that its child nodes draw themselves - in turn, each ViewGroup node in the tree is responsible for drawing its direct children.
As mentioned previously, each view group has the responsibility of measuring its available space, laying out its children, and calling `draw()` on each child to let it render itself. The children may request a size and location in the parent, but the parent object has the final decision on where how big each child can be.

### 3.14 Comparing Android UI Elements to Swing UI Elements.

As some developers who are reading this have probably coded UIs with Swing before here are some similarities between Android and Swing.

- **Activities** in Android refers *almost* to a *(J)Frame* in Swing.
- **Views** in Android refers to *(J)Components* in Swing.
- **TextViews** in Android refers to a *(J)Labels* in Swing.
- **EditTexts** in Android refers to a *(J)TextFields* in Swing.
- **Buttons** in Android refers to a *(J)Buttons* in Swing.

Setting listeners to a View is nearly the same in Android than in Swing.

```java
// Android
myView.setOnClickListener(new OnClickListener(){ ... })

// Swing
myButton.addActionListener(new ActionListener(){ ... })
```

### 3.15 The AndroidManifest.xml.

The *AndroidManifest.xml* is a required file for every Android application. It is located in the root folder of the application, and describes global values for your package, including the application components (activities, services, etc) that the package exposes to the ‘outer world’, what kind of data each of our Activities and co. can handle, and how they can be launched.

An important thing to mention of this file are its so called *IntentFilters*. These filters describe where and when that activity can be started. When an activity (or the operating system) wants to perform an action such as open a Web page or open a contact picker screen, it creates an *Intent* object. This Intent-object can hold several information describing what you want to do, what data is needed to accomplish it and other bits of information. Android compares the information in an Intent object with the intent filter exposed by every application and finds the activity most appropriate to handle the data or action specified by the caller. If there
it more than one application capable of handling that Intent, the user gets asked, which app he would prefer handling it.

Besides declaring your application's Activities, Content Providers, Services, and Intent Receivers, you can also specify permissions in AndroidManifest.xml.

### 3.16 General

A very simple AndroidManifest.xml looks like this:

```xml
<manifest version="1.0" encoding="utf-8">
  <package name="org.anddev.android.hello_android">
    <application>
      <activity name=".Hello_Anhroid">
        <intent-filter>
          <action name="android.intent.action.MAIN" />
          <category name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
    </application>
  </package>
</manifest>
```

- Almost every AndroidManifest.xml (as well as many other Android XML files) will include the namespace declaration (xmlns:android=\http://schemas.android.com/apk/res/android\) in its first element. This makes a variety of standard Android attributes available in the file, which will be used to supply most of the data for elements in that file.
- Almost every manifest includes a single <application> tag, which itself contains several tags describing Applications, IntentReceivers, etc... that are available in this application.
- If you want to make an Activity launchable directly through the user, you will need to make it support the MAIN action and LAUNCHER category.

What follows is a detailed list of the structure of an AndroidManifest file, describing all available <tags>, with an Example for each:

**<manifest>**

This is the root node of each AndroidManifest.xml. It contains the package-attribute, which points to any package in out Activity. Other Activities-path will base relative to its value.
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.anddev.android.smstretcher">

<uses-permission>
Describes a security permission, which your package must be granted in order for it to operate correctly (i.e. when you want to send SMS or use the Phone-Contacts). The permissions get granted by the user during installation of your application.

<uses-permission android:name="android.permission.RECEIVE_SMS"/>

<permission>
Declares a security permission that can be used to restrict which applications can access components or features in your (or another) package.

<permission/>

<instrumentation>
Declares the code of an instrumentation component that is available to test the functionality of this or another package. See Instrumentation for more details.

<instrumentation/>

<application>
Root element containing declarations of the application-level components contained in the package. This element can also include global and/or default attributes for the application, such as a label, icon, theme, required permission, etc.

<application android:icon="@drawable/icon"/>

<activity>
An Activity is the primary thing for an application to interact with the user. The initial screen the user sees when launching an application is an activity, and most other screens they use will be implemented as separate activities declared with additional activity tags.

<activity android:name=".Welcome" android:label="@string/app_name"/>
<intent-filter>
Declares what kind of Intents a component supports. In addition to the various kinds of values that can be specified under this element, attributes can be given here to supply a unique label, icon, and other information for the action being described.

<action
An action-type that the component supports.

<category
A category-type that the component supports.

<data
An MIME type, URI scheme, URI authority, or URI path that the component supports.

<meta-data
Adds a new piece of meta data to the activity, clients can retrieve through which ComponentInfo.metaData.

<receiver
An IntentReceiver allows an application to be told about changes to data or actions that happen, even if it is not currently running. As with the activity tag, you can optionally include 1+ <intent-filter> elements that the receiver supports or <meta-data> values, just all the same as with <activity>.

<service
A Service is a component that can run in the background for an arbitrary amount of time. As with the activity tag, you can optionally include one or more <intent-filter> elements that the service supports or <meta-data> values; see the activity's <intent-filter> and <meta-data> descriptions for more information.
<provider>
A ContentProvider is a component that manages persistent data and publishes it for access by other applications. You can also optionally attach one or more <meta-data> values, as described in the activity's <meta-data> description.

Of course all <tags> have to be </closed> or closed <directly/>.
Chapter 4

Arduino Development.

An Arduino is a single-board microcontroller and a software suite for programming it. The hardware consists of a simple open hardware design for the controller with an Atmel AVR processor and on-board I/O support. The software consists of a standard programming language and the boot loader that runs on the board.

Arduino hardware is programmed using a Wiring based language (syntax + libraries), similar to C++ with some simplifications and modifications, and a Processing based IDE.

Currently shipping versions can be purchased pre-assembled; hardware design information is available for those who would like to assemble an Arduino by hand. Additionally, Arduino-inspired clones with varying levels of compatibility have been released by third parties.

The Arduino project received an honorary mention in the Digital Communities category at the 2006 Prix Ars Electronica.

The project began in Ivrea, Italy in 2005 to make a device for controlling student-built interaction design projects less expensively than other prototyping systems available at the time. As of October 2008 more than 50,000 Arduino boards had been shipped. This has risen to more than 120,000 shipped boards as of February 2010.

4.1 Hardware

An Arduino board consists of an 8-bit Atmel AVR microcontroller with complementary components to facilitate programming and incorporation into other circuits. An important aspect of the Arduino is the standard way that connectors are exposed allowing the CPU board to be connected to a variety of interchangeable add-on modules (known as shields). Official Arduinos have used the megaAVR series of chips, specifically the ATmega8, ATmega168, ATmega328, and ATmega1280. A handful of other processors have been used by Arduino clones. Most boards include a 5-volt linear regulator and a 16 MHz crystal oscillator (or ceramic resonator in some variants), although some designs such as the LilyPad run at 8 MHz and dispense with the onboard voltage regulator due to specific form-factor restrictions. An Arduino's microcontroller is also pre-programmed with a bootloader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external chip programmer.
At a conceptual level, when using the Arduino software stack all boards are programmed over an RS-232 serial connection, but the way in which this is implemented varies by hardware version. Serial Arduino boards contain a simple inverter circuit to convert between RS-232-level and TTL-level signals. Current Arduino boards are programmed via USB, implemented using USB-to-serial adapter chips such as the FTDI FT232. Some variants, such as the Arduino Mini and the unofficial Boarduino, use a detachable USB-to-serial adapter board or cable, Bluetooth or other methods. (When used with traditional microcontroller tools instead of the Arduino IDE, standard AVR ISP programming is used.)

The Arduino board exposes most of the microcontroller's I/O pins for use by other circuits. The Diecimila, now superseded by the Duemilanove, for example, provides 14 digital I/O pins, 6 of which can produce PWM signals, and 6 analog inputs. These pins are available on the top of the board, via female 0.1 inch headers. Several plug-in application boards known as "shields" are also commercially available.

The Arduino Nano, and Arduino-compatible Barebones and Boarduino boards provide male header pins on the underside of the board to be plugged into solderless breadboards.

4.2 Software

The Arduino IDE is a cross-platform application written in Java which is derived from the IDE made for the Processing programming language and the Wiring project. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. There is typically no need to edit Makefiles or run programs on the command line.

The Arduino IDE comes with a C/C++ library called "Wiring" (from the project of the same name), which makes many common input/output operations much easier. Arduino programs are written in C/C++, although users only need to define two functions in order to make a runnable program:

- setup() – a function run once at the start of a program which can be used for initializing settings, and
- loop() – a function called repeatedly until the board is powered off.

A typical first program for a microcontroller is to simply blink a LED (light-emitting diode) on and off. In the Arduino environment, the user might write a program like this:

```
#define LED_PIN 13

void setup () {
  
```

24
The above code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the "Upload to I/O board" button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple main() function at the bottom, to make it a valid C++ program. The Arduino IDE uses the GNU toolchain and AVR Libc to compile programs, and uses avrdude to upload programs to the board.

Arduino BT is the board used in our application due to the bluetooth communication.

### 4.3 Arduino BT

The Arduino BT is a microcontroller board based on the ATmega168 and the Bluegiga WT11 bluetooth module. It supports wireless serial communication over bluetooth (but is not compatible with Bluetooth headsets or other audio devices). It has 14 digital input/output pins (of which 6 can be used as PWM outputs and one can be used to reset the WT11 module), 6 analog inputs, a 16 MHz crystal oscillator, screw terminals for power, an ICSP header, and a reset button. It contains everything needed to support the microcontroller and can be programmed wirelessly over the Bluetooth connection.

![Arduino BT board](image-url)
4.4 Summary

- **Microcontroller**: ATmega168
- **Operating Voltage**: 5V
- **Input Voltage**: 1.2-5.5 V
- **Digital I/O Pins**: 14 (of which 6 provide PWM output)
- **Analog Input Pins**: 6
- **DC Current per I/O Pin**: 40 mA
- **DC Current for 3.3V Pin**: 50 mA
- **Flash Memory**: 16 KB (of which 2 KB used by bootloader)
- **SRAM**: 1 KB
- **EEPROM**: 512 bytes
- **Clock Speed**: 16 MHz

4.5 Schematic

![Arduino BT Schematic](image)

Figure 4.2 - Arduino BT schematic.
4.6 Power

The Arduino BT can be powered via the V+ and GND screw terminals. The board contains a DC-DC convector that allows it to be powered with as little as 1.2V, but a maximum of 5.5V. Higher voltages or reversed polarity in the power supply can damage or destroy the board.

The power pins are as follows:

- **9V.** The input voltage to the Arduino board (i.e. the same as the V+ screw terminal). You can supply voltage through this pin, or, if supplying voltage via the screw terminals, access it through this pin. **Warning:** despite the label, do not attach 9V to this pin. It will damage the board.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from V+ via the on-board DC-DC convertor, or be supplied by a regulated 5V supply.
- **GND.** Ground pins.

4.7 Input and Output

Each of the 14 digital pins on the BT can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial:** 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the Bluegiga WT11 module.
- **External Interrupts:** 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM:** 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- **SPI:** 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.
- **BT Reset:** 7. Connected to the reset line of the Bluegiga WT11 module, which is active high.
• LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The BT has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and some low-level code. Additionally, some pins have specialized functionality:

• I2C: 4 (SDA) and 5 (SCL). Support I2C (TWI) communication using the Wire library

There are a couple of other pins on the board:

• AREF. Reference voltage for the analog inputs. Used with analogReference().

4.8 Bluetooth Communication

The Bluegiga WT11 module on the Arduino BT provides Bluetooth communication with computers, phones, and other Bluetooth devices. The WT11 communicates with the ATmega168 via serial (shared with the RX and TX pins on the board). It comes configured for 115200 baud communication. The module should be configurable and detectable by your operating system's bluetooth drivers, which should then provide a virtual com port for use by other applications. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board over this bluetooth connection. The board can also be reprogrammed using this same wireless connection.

The WT11 is specially configured for use in the Arduino BT. Its name is set to ARDUINOBT and passcode to 12345.

4.9 Communication

The Arduino BT has a number of other facilities for communicating. The ATmega168's UART TTL (5V) serial communication is available on digital pins 0 (RX) and 1 (TX) as well as being connected to the WT11 module.

A SoftwareSerial library allows for serial communication on any of the BT's digital pins.

The ATmega168 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus.
4.10 Information about the Arduino BT

In most respects, the Arduino BT is similar to the Arduino Diecimila. Here are the main differences of BT board (besides the fact that it communicates over bluetooth instead of USB):

- The Arduino BT is more fragile and easy to break than a regular Arduino board.
- Don’t power the board with more than 5.5 volts to the or reverse the polarity (power and ground pins) of your power supply, or you might kill the ATmega168 on the Arduino BT. The Arduino BT can, however, run with a minimum of 1.2 volts, making it easier to power with batteries.
- The microcontroller (an ATmega168) on the Arduino BT is a physically smaller version of the chip on the USB Arduino boards. You can’t remove it, so if you kill it, you need a new Arduino BT.
- There are two extra analog inputs on the Arduino BT (8 total). Two of these, however, are not connected to the pin headers on the board; you’ll need to solder something to the pads next to the numbers "6" and "7".
- Pin 7 is connected to the reset pin of the bluetooth module; don’t use it for anything (except resetting the module).

4.11 Using the Arduino BT

The on-board serial communication between the bluetooth module and the Arduino sketch (running on the ATmega168) needs to be at 115200 baud (i.e. call Serial.begin(115200) in your setup() function). Communication between the bluetooth module and the computer can be at any baud rate.

Communication between the BT module and the computer can be temperamental. You might want to open the serial monitor a couple of seconds after resetting the board.
Chapter 5

Amarino, “Android meets Arduino”.

Normally smartphone events are tightly coupled to your phone device itself. When your cell phone is ringing, your phone speaker plays a ringtone. When you get a new text message, your phone displays it on its screen. Wouldn't it be thrilling to make those phone events visible somewhere else, on your wearable, in your living room, on your robot, in your office or wherever you want it to occur? Or would you like to use your smartphone sensors, like the accelerometer, light sensor, compass or your touchscreen to control other devices? 'Android meets arduino' is a toolkit, basically consisting of an Android application and an Arduino library which will help you to interface with your phone in a new dimension. You can build your own interfaces almost without any programming experience.

Figure 5.1 - Amarino Logo.

5.1 Documentation

Amarino basically consists of three main parts (two mandatory and one optional):

- Android application called "Amarino"
- Arduino library called "MeetAndroid"
- Amarino Plug-In Bundle (optional)
If you want to work with Amarino you need at least:

- An Android phone (Android 2.x works best, however Android 1.6 is also supported but not all models will work, e.g. HTC Hero and Samsung Behold II with Android 1.x won't work)
- An Arduino board.
- A Bluetooth shield for your Arduino.

It's important to set your Bluetooth module to 57600 baud (using Putty or other Terminal programs) otherwise you will get corrupted data. Only the Arduino BT board works with 115200 baud without flaws.

5.2 Download.


- **Amarino - Android Application**
  Install Amarino_2.apk on your phone. You can do this by calling "adb install Amarino_2.apk" from a console. Make sure that your phone is connected to your computer and the Android SDK Tools directory is added to your PATH.

- **MeetAndroid - Arduino Library**
  The MeetAndroid library (MeetAndroid.zip) needs to be extracted into your Arduino sketches directory(your_arduino_sketches_dir/libraries/MeetAndroid). Restart your Arduino IDE, if everything went well, the MeetAndroid library can be found under "Sketch->Import Library..."

- **Amarino Plug-in Bundle**
  Amarino incorporates a plug-in mechanism which allows developers to integrate their own events into Amarino. We provide some useful plug-ins to start with. Install the AmarinoPluginBundle.apk onto your phone and you will find many plug-ins (events) added to Amarino.

- **Amarino Library**
  Amarino has also an Application Programming Interface (API) which you can use to send and receive data to Arduino within your own application using Amarino's powerful communication infrastructure. There is a JavaDoc about this library at Documentation providing all necessary information you need to use the library. The SensorGraph example below might also help to understand the API.
5.3 Connecting Android to Arduino.

This chapter assumes that you have installed Amarino on your phone and you have added the MeetAndroid library to your Arduino working environment and your Bluetooth module is set to 57600 baud. Note that only Arduino BT boards will work with 115200 baud.

5.4 Android setup.

Follow the steps below to make the connection:

1 -

Figure 5.2 - Amarino’s first screen.

This is the first screen you will see when you start Amarino on your phone. Since Amarino is all about connecting your phone to an Arduino, the very first step you have to do is to search for your Arduino Bluetooth device you want to talk to. To do that hit the "Add BT Device" button and wait until your Arduino Bluetooth module pops up. If it will not show up even if the discover process has already finished, you should check if your Arduino Bluetooth module is powered and discoverable.
Maybe you will find more than one device, because of other Bluetooth devices around you. You need to find out which one is your Arduino Bluetooth module. You found it, then select it to add it to Amarino.

If you managed to add your BT device to Amarino you will see one new device, namely the device you added, with a connect button next to it at the main screen of your Amarino application.
One thing you have to know before we proceed is, that Bluetooth devices need to become acquainted to each other (authenticated) before they can communicate. A process called "pairing" or "bonding", I am rather sure you heart of that.

Finally it is time to hit the connect button. Don't worry you will get a "Connection failed" message or a message telling you to pull down the status bar to enter the pin, but that is just fine. We tried to connect to an unpaired device, which is as I said not possible, and now you proved it.

But apart from that, it was necessary to get the pairing request notification in your notification bar (on the very top of your phone). Pull down the notification bar as shown in the screenshot and you will see the "Pairing request" notification.

Figure 5.5 - Pairing request.

Figure 5.6 - Introducing bluetooth PIN.
Tap on the "Pairing request" notification and the Bluetooth pairing dialog pops up. This is where you enter your Bluetooth module's super high confidential top secret PIN. Almost always it is "1234" or for ArduinoBT boards "12345". Some modules will have "0000" but less common. Type it in and hit "OK". If the PIN was right, pairing is done.

Hit the connect button once again. This time it should connect successfully indicated by a green light on your Bluetooth module and a green light right above the connect button. If not, repeat the pairing process.

This means basic connection works. Before you proceed, hit the "Disconnect" button to disconnect.
Tap on the left red cabinet icon to manage events which should be sent to your Arduino. Select "Add Event" and search for "Test Event" in the upcoming list. Select "Test Event". A description page will be opened telling you what Test Event is all about. For now simply say "Save". This will add the Test Event which sends random data to Arduino as soon as you establish a connection.

5.5 Arduino setup.
Open your Arduino working environment and select "File->Examples->MeetAndroid->Test".

Change the baud rate of your Bluetooth module to 57600 baud, especially if you do not use an Arduino BT board with built-in Bluetooth (refer to your Bluetooth module documentation about how to change the baud rate). When you buy a Bluetooth Mate or BlueSMiRF Gold Bluetooth module, normally the baud rate is set to 9600 or 115200. But this doesn't work well for most Arduinos. Set it to 57600 baud and make sure the baud rate of your Bluetooth module matches the baud rate in your Arduino sketch.

Upload the sketch to your Arduino. If your Arduino has no onboard LED connected to pin 13 you might put your own LED to pin 13 of your Arduino. You should already know how this works. Otherwise you should look it up on the Arduino website.

Your phone is now set up, your Arduino is loaded with a test program. So far so good. Now comes the great moment. Let's see if they understand each other.

Power on your Arduino with your Bluetooth shield attached to it. Take your Android phone and go to the main screen. Hit the connect button next to your added BT device. It should start connecting. If your connection could be established the indicator changes to green.

If everything went right your led on pin 13 will blink every 5 seconds for 1 second.
5.6 Monitoring.

To see what is going on in the background just hit the monitoring button and look at the communication process.

5.7 Real-time data.

The event management module also gives you some feedback providing real-time data to you. Instead of monitoring go back to the event module (red cabinet icon) and see which random values are sent from the Test event. Real-time data are only visible if a connection is up and running. However you can force enable events to show their data without being connected (long press on an added event to get a context menu with options to force enable/disable).
Chapter 6

Conclusions

In this project we have succeeded in creating an application with various aspects that are currently under development due to the constant developments of mobile technologies.

The main achievement of this project is that we have managed to transmit data between devices via bluetooth android and a separate device connected wirelessly, in our case, an analog sensor connected to an Arduino board.

This is something important because the connection between Arduino and Android has been developed only once, in particular by a group called Amarino from MIT which is led by Bonifaz Kaufmann.

In addition, the project has helped us a lot to learn how develop in a new platform and unknown to us and how to approach the problems that arise during the process.

I think the hardest part and that gives the importance of this project has been the research work.

It is an honor for us if this report could potentially serve as a tutorial to someone to learn how to involve bluetooth technology to Android.
Appendix

Source Code.

A.1 AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.PFC.inpress"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".main"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".screen1"
            android:label="@string/app_name">
```
A.2 /inPress/src/com.PFC.inpress

A.2.1 about.java

```java
package com.PFC.inpress;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class about extends Activity implements OnClickListener{

private Button back;

public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.about);
back = (Button) findViewById(R.id.back);
back.setOnClickListener(this);
}

public void onClick(View v) {
if(v.equals(back)){
finish();
}
}
}
```

A.2.2 info.java

```java
package com.PFC.inpress;

public class info {
```
private double latitude;
private double longitude;
private String name;

private static info INSTANCE = null;

private info() {}

private synchronized static void createInstance() {
  if (INSTANCE == null) {
    INSTANCE = new info();
  }
}

public static info getInstance() {
  if (INSTANCE == null) createInstance();
  return INSTANCE;
}

public void set_latitude(double latitude){
  this.latitude=latitude;
}

public double get_latitude(){
  return this.latitude;
}

public void set_longitude(double longitude){
  this.longitude=longitude;
}

public double get_longitude(){
  return this.longitude;
}

public void set_name(String name){
  this.name=name;
}

public String get_name(){
  return this.name;
}

A.2.3 LevelView.java

package com.PFC.inpress;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.drawable.Drawable;
import android.util.AttributeSet;
import android.view.View;

class LevelView extends View {
  private Drawable mGreen;
  private Drawable mRed;
  private Drawable mBlue;
  private Drawable mYellow;
  private Drawable mWhite;
  private Paint mBackgroundColorPaint;
  private int mHeight;
  private int mWidth;
  private int mThreshold = 0;
  private int mVol = 0;
  private int mMinimo = 0;

  public LevelView(Context context, AttributeSet attrs) {
    super(context, attrs);
    mGreen = context.getResources().getDrawable(
R.drawable.greenbar);
mRed = context.getResources().getDrawable(
R.drawable.redbar);
mBlue = context.getResources().getDrawable(
R.drawable.bluebar);
mWhite = context.getResources().getDrawable(
R.drawable.whitebar);
mYellow = context.getResources().getDrawable(
R.drawable.yellowbar);
mHeight = mGreen.getIntrinsicHeight();
setMinimumHeight(mHeight*300);
mWidth = mGreen.getIntrinsicWidth();
setMinimumWidth(mWidth);
mBackgroundPaint = new Paint();
mBackgroundPaint.setColor(Color.BLACK);
}

public void setLevel(int volume, int threshold, int minimo) {
    if (volume == mVol && threshold == mThreshold && minimo==mMinimo)
        return;
    mVol = volume;
    mThreshold = threshold;
    mMinimo = minimo;
    invalidate();
}

public void onDraw(Canvas canvas) {
    canvas.drawPaint(mBackgroundPaint);
    for (int i=0; i<= mVol; i++) {
        Drawable bar;
        if (i<mThreshold && i>mMinimo) bar = mGreen;
        else if(i>mThreshold && i>((mThreshold-mMinimo)*2)+mMinimo) bar = mYellow;
        else if(i>=(mThreshold-mMinimo)*2+mMinimo) bar = mWhite;
        else if(i>=mThreshold) bar = mRed;
        else bar = mBlue;
        bar.setBounds(0, ((mThreshold)*2-i), mWidth, ((mThreshold)*2-i+1));
        bar.draw(canvas);
    }
}

A.2.4 LevelView2.java

package com.PFC.inpress;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.drawable.Drawable;
import android.util.AttributeSet;
import android.view.View;

public class LevelView2 extends View {
    private Drawable mGreen;
    private Drawable mRed;
    private Drawable mYellow;
    private Paint mBackgroundPaint;
    private int mHeight;
    private int mWidth;
    private int mThreshold = 0;
    private int mVol = 0;
    private int mMinimo = 0;

    public LevelView2(Context context, AttributeSet attrs) {
        super(context, attrs);
        mGreen = context.getResources().getDrawable(43...
R.drawable.greenbar2);
mRed = context.getResources().getDrawable(R.drawable.redbar2);
mYellow = context.getResources().getDrawable(R.drawable.yellowbar2);
mHeight = mGreen.getIntrinsicHeight();
setMinimumHeight(mHeight*1000);
mWidth = mGreen.getIntrinsicWidth();
setMinimumWidth(mWidth);
mBackgroundPaint = new Paint();
mBackgroundPaint.setColor(Color.BLACK);
}
public void setLevel(int volume, int threshold, int minimo) {
    if (volume == mVol && threshold == mThreshold && minimo==mMinimo) return;
    mVol = volume;
mThreshold = threshold;
mMinimo = minimo;
    invalidate();
}
public void onDraw(Canvas canvas) {
    canvas.drawPaint(mBackgroundPaint);
    for (int i=minimo+1; i<= mVol; i++) {
        Drawable bar;
        if (i<threshold && i>minimo) {
            bar = mGreen;
            bar.setBounds(0, ((threshold-minimo)*2+minimo-i)*7, mWidth,
            ((threshold-minimo)*2+minimo-i+1)*7);
            bar.draw(canvas);
        } else if(i==threshold) {
            bar = mRed;
            bar.setBounds(0, ((threshold-minimo)*2+minimo-i)*7, mWidth,
            ((threshold-minimo)*2+minimo-i+1)*7);
            bar.draw(canvas);
        } else if(i>threshold && i<(threshold-minimo)*2+minimo) {
            bar = mYellow;
            bar.setBounds(0, ((threshold-minimo)*2+minimo-i)*7, mWidth,
            ((threshold-minimo)*2+minimo-i+1)*7);
            bar.draw(canvas);
        } 
    }
}

A.2.5 main.java

package com.PFC.inpress;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
public class main extends Activity implements OnClickListener {
    private Button start;
    private Button exit;
    private Button about;
    private EditText paciente;
    private String name;
    info DATOS = info.getInstance();
}
private int GET_CODE=0;

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    start = (Button) findViewById(R.id.start);
    start.setOnClickListener(this);
    paciente = (EditText) findViewById(R.id.paciente);
    exit = (Button) findViewById(R.id.exit);
    exit.setOnClickListener(this);
    about = (Button) findViewById(R.id.about);
    about.setOnClickListener(this);
}

public void onClick(View v) {
    if(v.equals(start)){
        name = paciente.getText().toString();
        DATOS.set_name(name);
        Intent intent = new Intent(main.this,screen1.class);
        startActivityForResult(intent,GET_CODE);
    }
    if(v.equals(exit)){
        finish();
    }
    if(v.equals(about)){
        Intent intent = new Intent(main.this,about.class);
        startActivityForResult(intent,GET_CODE);
    }
}

A.2.6 screen1.java

package com.PFC.inpress;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class screen1 extends Activity implements OnClickListener {
    private int GET_CODE=0;
    private Button SETUP;
    private Button back;
    private TextView texto;

    // GPS
    private ProgressDialog pd;
    info DATOS = info.getInstance();
    LocationManager mLocationManager;
Location mLocation;
MyLocationListener mLocationListener;
Location currentLocation = null;
// GPS END

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.screen1);
    SETUP = (Button) findViewById(R.id.SETUP);
    SETUP.setOnClickListener(this);
    back = (Button) findViewById(R.id.back);
    back.setOnClickListener(this);
    texto = (TextView) findViewById(R.id.texto);
    writeSignalGPS();
    texto.append("\n");
    texto.append("\n");
    texto.append("\n");
    texto.append("1- Press the sensor firmly."");
    texto.append("\n");
    texto.append("\n");
    texto.append("2- Click SETUP and hold the sensor until the program tells.");
    texto.append("\n");
    texto.append("\n");
}

public void onClick(View v) {
    if(v.equals(SETUP)){
        Intent intent = new Intent(screen1.this,screen2.class);
        startActivityForResult(intent,GET_CODE);
    } else if(v.equals(back)){
        finish();
    }
}

private Runnable mRunnable2 = new Runnable() {
    public void run() {
        mLocationManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
        if (mLocationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
            Looper.prepare();
            mLocationListener = new MyLocationListener();
            mLocationManager.requestLocationUpdates(
                    LocationManager.GPS_PROVIDER, 0, 0, mLocationListener);
            Looper.loop();
            Looper.myLooper().quit();
        } else {
            Toast.makeText(getBaseContext(), getResources().getString(R.string.gps_signal_not_found), Toast.LENGTH_LONG).show();
        }
    }
};
private void setCurrentLocation(Location loc) {
    currentLocation = loc;
}
private void writeSignalGPS() {
    DialogInterface.OnCancelListener dialogCancel = new
    DialogInterface.OnCancelListener() {
        public void onCancel(DialogInterface dialog) {
            Toast.makeText(getBaseContext(),
                    getResources().getString(R.string.gps_signal_not_found),
                    Toast.LENGTH_LONG).show();
            handler.sendEmptyMessage(0);
        }
    };
    pd = ProgressDialog.show(this, this.getResources().getString(R.string.search),
            (dialog, which) -> dialogCancel.onCancel(dialog)) { (dialog, which) -> dialogCancel.onCancel(dialog)}
            public void onCancel(DialogInterface dialog) {
                Toast.makeText(getBaseContext(),
                        getResources().getString(R.string.gps_signal_not_found),
                        Toast.LENGTH_LONG).show();
                handler.sendEmptyMessage(0);
            };
}
this.getResources().getString(R.string.search_signal_gps),
true, true, dialogCancel);
mThread = new Thread(mRunnable2);
mThread.start();
}
private Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        pd.dismiss();
        locationManager.removeUpdates(locationListener);
        if (currentLocation != null) {
            texto.append("\n");
            texto.append("\n");
            texto.append("Latitude: " + currentLocation.getLatitude());
            texto.append("\n");
            texto.append("Longitude: " + currentLocation.getLongitude());
            DATOS.set_latitude(currentLocation.getLatitude());
            DATOS.set_longitude(currentLocation.getLongitude());
        }
    }
}
private class MyLocationListener implements LocationListener {
    @Override
    public void onLocationChanged(Location loc) {
        if (loc != null) {
            Toast.makeText(getBaseContext(),
                    getResources().getString(R.string.gps_signal_found),
                    Toast.LENGTH_LONG).show();
            setCurrentLocation(loc);
            handler.sendEmptyMessage(0);
        }
    }
    @Override
    public void onProviderDisabled(String provider) {
        // TODO Auto-generated method stub
    }
    @Override
    public void onProviderEnabled(String provider) {
        // TODO Auto-generated method stub
    }
    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
        // TODO Auto-generated method stub
    }
}

A.2.7 screen2.java

package com.PFC.inpress;

import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.util.Date;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import at.abraxas.amarino.Amarino;
import at.abraxas.amarino.AmarinoIntent;

public class screen2 extends Activity implements OnClickListener{

    private static final String TAG = "SensorGraph";

    // change this to your Bluetooth device address
    private static final String DEVICE_ADDRESS = "00:06:66:03:17:17"; //"00:06:66:03:73:7B";
    private LevelView mSound;
    private LevelView2 mSound2;
    private TextView mValueTV;
    private TextView texto;

    private int i=0,j=0,k=0,aux=0;
    private int max1=0,max2=0,max3=0,maximo=0,threshold;
    private int min1=10000,min2=10000,min3=10000;
    private int minimo=0;

    private int num;

    private int sensorReading=0;

    private Button EXIT;
    private Button START;

    Date date = new Date();
    FileInputStream fin;
    FileOutputStream fon_conf;
    FileOutputStream fon_data;
    OutputStreamWriter osw = null;
    OutputStreamWriter osw2 = null;
}
DataOutputStream dos;
DataOutputStream dos2;

info INFO = info.getInstance();

String CONF;
String DAT;

private ArduinoReceiver arduinoReceiver = new ArduinoReceiver();

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.screen2);

    // get handles to Views defined in our layout file
    texto = (TextView) findViewById(R.id.texto);
    mValueTV = (TextView) findViewById(R.id.value);
    aux=0;
    mSound = (LevelView) findViewById(R.id.volume);
    mSound2 = (LevelView2) findViewById(R.id.volume2);

    EXIT = (Button) findViewById(R.id.exit);
    EXIT.setOnClickListener(this);
    START = (Button) findViewById(R.id.start);
    START.setOnClickListener(this);
    try {
        fin = new FileInputStream("/sdcard/datos.txt");
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    try {
        CONF="CONF_"+INFO.get_name().toString()+".txt"
        DAT="DATA_"+INFO.get_name().toString()+".txt"
        fon_conf = openFileOutput("CONF_"+INFO.get_name().toString()
".txt",MODE_WORLD_WRITEABLE|MODE_WORLD_READABLE);
        fon_data = openFileOutput("DATA_"+INFO.get_name().toString()
".txt",MODE_WORLD_WRITEABLE|MODE_WORLD_READABLE);
        osw = new OutputStreamWriter(fon_conf);
        osw2 = new OutputStreamWriter(fon_data);
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    File file = new File(Environment.getExternalStorageDirectory().getAbsolutePath() + "/"+CONF.toString());
    File file2 = new File(Environment.getExternalStorageDirectory().getAbsolutePath() + "/"+DAT.toString());
// Delete any previous recording.
if (file.exists())
    file.delete();

if (file2.exists())
    file2.delete();

// Create the new file.
try {
    file.createNewFile();
} catch (IOException e) {
    throw new IllegalStateException("Failed to create " + file.toString());
}

try {
    file2.createNewFile();
} catch (IOException e) {
    throw new IllegalStateException("Failed to create " + file2.toString());
}

try {
    try {
        OutputStream os = new FileOutputStream(file);
        BufferedOutputStream bos = new BufferedOutputStream(os);
        dos = new DataOutputStream(bos);
        dos.writeChars(date.toGMTString());
        dos.writeChar('n');
        dos.writeChars(INFO.get_name().toString());
        dos.writeChar('n');
        dos.writeChars("Longitude: " + INFO.get_longitude());
        dos.writeChar('n');
        dos.writeChars("Latitude: " + INFO.get_latitude());
        dos.writeChar('n');
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    try {
        OutputStream os2 = new FileOutputStream(file2);
        BufferedOutputStream bos2 = new BufferedOutputStream(os2);
        dos2 = new DataOutputStream(bos2);
        dos2.writeChars(INFO.get_name().toString());
        dos2.writeChar('n');
        dos2.writeChars(date.toGMTString());
        dos2.writeChar('n');
        dos2.writeChars("Longitude: " + INFO.get_longitude());
        dos2.writeChar('n');
        dos2.writeChars("Latitude: " + INFO.get_latitude());
        dos2.writeChar('n');
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

50
public void onClick(View v) {

    if(v.equals(EXIT)) {

        try {
            osw.close();
            osw2.close();
            dos.flush();
            dos2.flush();
            dos.close();
            dos2.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        Intent emailIntent = new Intent
                (android.content.Intent.ACTION_SEND);
        String[] recipients = new String[] { "vandiasierra@gmail.com", ""};
        emailIntent.putExtra(android.content.Intent.EXTRA_EMAIL, recipients);
        emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("file:///sdcard/"+CONF.toString()));
        emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, "Configuration File of " + INFO.get_name().toString());
        emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Configuration File of " + INFO.get_name().toString());
        startActivity(Intent.createChooser(emailIntent, "Send mail...");

        Intent emailIntent2 = new Intent
                (android.content.Intent.ACTION_SEND);
        recipients = new String[] { "vandiasierra@gmail.com", ""};
        emailIntent2.putExtra(Intent.EXTRA_EMAIL, recipients);
        emailIntent2.putExtra(Intent.EXTRA_STREAM, Uri.parse("file:///sdcard/"+DAT.toString()));
        emailIntent2.putExtra(Intent.EXTRA_SUBJECT, "Data File of " + INFO.get_name().toString());
        emailIntent2.putExtra(Intent.EXTRA_SUBJECT, "Data File of " + INFO.get_name().toString());
        startActivity(Intent.createChooser(emailIntent2, "Send mail...");
        finish();
    }

    // THIS IS SIMULATION. WITHOUT ARDUINO, USING FILES !!!!
/*if(v.equals(START)){

}}
try {
   // Open an input stream

   String caract="t";

   try{
      if( caract!=null ){
         caract = new DataInputStream(fin).readLine();
         if(caract!=null){
            num = Integer.parseInt(caract);
            Log.i("--EARL--", "num "+num + " read");
            mValueTV.setText(""+num);
            if(i<20){
               texto.setText("SETTING UP ... ");
               i++;
               osw.write(caract);
               osw.write('
');
               dos.writeChars(caract);
               dos.writeChar('
');
               if(num>max1){
                  max1=num;
               }
               if(num<min1){
                  min1=num;
               }
            }
            if(i==20 && j<20){
               j++;
               osw.write(caract);
               osw.write('
');
               dos.writeChars(caract);
               dos.writeChar('
');
               if(num>max2){
                  max2=num;
               }
               if(num<min2){
                  min2=num;
               }
            }
            if(i==20 && j==20 && k<20){
               k++;
               //fon_conf.write(num);
            }
         }
      }
   }
}
```java
osw.write(caract);
osw.write('n');
dos.writeChars(caract);
dos.writeChar('n');

if(num>max3){
    max3=num;
}
if(num<min3){
    min3=num;
}

if(i==20 && j==20 && k==20){
    maximo=(max1+max2+max3)/3;
    minimo=(min1+min2+min3)/3;
    osw2.write(caract);
    osw2.write('n');
    dos2.writeChars(caract);
    dos2.writeChar('n');
    threshold=maximo/20; // umbral a 5%
    int thresholdmin=((maximo-minimo)/20)+minimo;
    int thresholdreal=(maximo-minimo)/20;
    int topbar=thresholdreal+minimo;
    int Threshold2=(thresholdmin-
    // para probar sensorReading/15
    // mSound.setLevel
    (sensorReading,thresholdmin,minimo);
    (num,thresholdmin,minimo);
    (num,thresholdmin,minimo);
    mValueTV.setBackgroundColor(0xffffffff);
    num<Threshold2){
        mValueTV.setBackgroundColor(0xffffffff00);
    mValueTV.setBackgroundColor(0xffffff0000);
    num > minimo){
```
mValueTV.setBackgroundColor(0xff00ff00);

mValueTV.setBackgroundColor(0xff0000ff);

LA VARIACION
20,threshold);
{
  RED COLOR\n\n"; texto.setText("TRY TO GET

"+INFO.get_name.toString());

());

"+INFO.get_longitude());

"+INFO.get_latitude());

= "+maximo);

= "+minimo);

"+thresholdmin);

"+Threshold2);

AlertDialog.Builder(screen2.this)
configured correctly.\nSet the bar at the threshold.")
DialogInterface.OnClickListener() {
(DialogInterface dialog, int what) {
will close on its own

//EL DE ABAJO SIRVE PARA COMPROBAR
//mSound.setLevel(sensorReading/
if(aux==0)

texto.setText("\t\tTRY TO GET
texto.append(""+INFO.get_name
texto.append("\n
texto.append(""+date.toString

texto.append("\n
texto.append("Longitude =
texto.append("\n
texto.append("\n
texto.append("Latitude =
texto.append("\n
texto.append("\n
texto.append("\n
texto.append("\n
texto.append("\n
texto.append("\n
texto.append("Threshold =
texto.append("\n
texto.append("\n
aux=1;

AlertDialog builder =new
.setTitle("SETUP")
.setMessage("The system is
public void onClick

  // do nothing ,ÅA it
}
)
.show();
int num=new DataInputStream(fin).readInt();

} catch (EOFException e) {
    Log.e("fin","fin");
    fin.close();
}

// Catches any error conditions
catch (IOException e) {
    Log.e("no se lee","no se lee");
}

}@Override
protected void onStart() {
    super.onStart();
    // in order to receive broadcasted intents we need to register our receiver
    registerReceiver(arduinoReceiver, new IntentFilter(AmarinoIntent.ACTION_RECEIVED));
    // this is how you tell Amarino to connect to a specific BT device from
    within your own code
    Amarino.connect(this, DEVICE_ADDRESS);
}

@Override
protected void onStop() {
    super.onStop();
    // if you connect in onStart() you must not forget to disconnect when your
    app is closed
    Amarino.disconnect(this, DEVICE_ADDRESS);
    // do never forget to unregister a registered receiver
    unregisterReceiver(arduinoReceiver);
}

//DECOMENTAR PARA USAR CON ARDUINO
public class ArduinoReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String data = null;

        // the device address from which the data was sent, we don't need it here but to demonstrate how you retrieve it
        final String address = intent.getStringExtra(AmarinoIntent.EXTRA_DEVICE_ADDRESS);

        // the type of data which is added to the intent
        final int dataType = intent.getIntExtra(AmarinoIntent.EXTRA_DATA_TYPE, -1);

        // we only expect String data though, but it is better to check if really string was sent
        // later Amarino will support different data types, so far data comes always as string and
        // you have to parse the data to the type you have sent from Arduino, like it is shown below
        if (dataType == AmarinoIntent.STRING_EXTRA) {
            data = intent.getStringExtra(AmarinoIntent.EXTRA_DATA);
            if (data != null) {
                mValueTV.setText(data);
                try {
                    // since we know that our string value is an int number we can parse it to an integer
                    final int num = Integer.parseInt(data);

                    if (i < 20) {
                        texto.setText("SETTING UP ...
                        ");
                        i++;
                        try {
                            osw.write(data);
                            osw.write("\n");
                            dos.writeChars(data);
                            dos.writeChar("\n");
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                    if (num > max1) {
                        max1 = num;
                    }
                    if (num < min1) {
                        min1 = num;
                    }
                }
            }
            if (i == 20 && j < 20) {
                // Y CORREGIR LAS COSAS TAL COMO com.ivasdroid.leer INDICA !!!!!!!!
            }
        }
    }
}

56
j++; 
    try {
        osw.write(data);
        osw.write('\n');
        dos.writeChars(data);
        dos.writeChar('\n');
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    if(num>max2){
        max2=num;
    }
    if(num<min2){
        min2=num;
    }
}
if(i==20 && j==20 && k<20){
k++; 
    try {
        osw.write(data);
        osw.write('\n');
        dos.writeChars(data);
        dos.writeChar('\n');
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    if(num>max3){
        max3=num;
    }
    if(num<min3){
        min3=num;
    }
}
if(i==20 && j==20 && k==20)
{
    maximo=(max1+max2+max3)/3;
    minimo=(min1+min2+min3)/3;
    try {
        osw2.write(data);
        osw2.write('\n');
        dos2.writeChars(data);
        dos2.writeChar('\n');
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
threshold = maximo/20; // umbral a 5%
int thresholdmin = ((maximo-minimo)/20+minimo);
int thresholdreal = (maximo-minimo)/20;
int tobar = thresholdreal + minimo;
int Threshold2 = (thresholdmin -

// para probar sensorReading/15
// mSound.setLevel
mSound.setLevel
mSound2.setLevel

if (num >= Threshold2) {
}
else if (num > thresholdmin &&

}
else if (num == thresholdmin &&

}
else if (num < thresholdmin &&

}
else

mValueTV.setBackgroundColor(0xffffffff);

if (aux == 0) {
    texto.setText("TRY TO GET RED COLOR

    texto.append("" + INFO.get_name().toString());
    texto.append("n");
    texto.append("n");
}

else
    mValueTV.setBackgroundColor(0xff00ff00);

else
    mValueTV.setBackgroundColor(0xff0000ff);

// EL DE ABAJO SIRVE PARA COMPROBAR
// mSound.setLevel(sensorReading/15)

mValueTV.setBackgroundColor(0x000000);

mValueTV.setBackgroundColor(0x00ff00);

mValueTV.setBackgroundColor(0x0000ff);

mValueTV.setBackgroundColor(0xffff0000);

mValueTV.setBackgroundColor(0xffffff00);

mValueTV.setBackgroundColor(0xffffffff);

mValueTV.setBackgroundColor(0xffff0000);

mValueTV.setBackgroundColor(0xffffff00);

mValueTV.setBackgroundColor(0xff00ff00);

mValueTV.setBackgroundColor(0xff0000ff);

// EL DE ABAJO SIRVE PARA COMPROBAR
// mSound.setLevel(sensorReading/15)

if (aux == 0) {
    texto.setText("TRY TO GET RED COLOR

    texto.append("" + INFO.get_name().toString());
    texto.append("n");
    texto.append("n");
}

else
    mValueTV.setBackgroundColor(0xff00ff00);

else
    mValueTV.setBackgroundColor(0xff0000ff);

// EL DE ABAJO SIRVE PARA COMPROBAR
// mSound.setLevel(sensorReading/15)

if (aux == 0) {
    texto.setText("TRY TO GET RED COLOR

    texto.append("" + INFO.get_name().toString());
    texto.append("n");
    texto.append("n");
}

else
    mValueTV.setBackgroundColor(0xff00ff00);

else
    mValueTV.setBackgroundColor(0xff0000ff);
texto.append("Longitude = 
\nlatitude = 
\nMedium maximum = 
\nMedium minimum = 
\nThreshold = 
\n2 Threshold = 
\nAux=1;

AlertDialog builder = new AlertDialog.Builder(screen2.this) 
 .setTitle("SETUP") 
 .setMessage("The system is configured correctly. Set the bar at the threshold.") 
 .setNeutralButton("OK", new 
 public void onClick 
 // do nothing, it will close on its own 
 )
 .show();

} 

\n\nA.3
/inPress/res/layout

A.3.1 about.xml

<ScrollView
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#fff">
  <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <ImageView
        android:id="@+id/fbf"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/fatebenefratelli"/>
    <TextView
        android:id="@+id/texto3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/b"
        android:textSize="15sp"
        android:textStyle="bold"
        android:gravity="center"/>
    <Button
        android:id="@+id/back"
        android:text="@string/back"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:textStyle="normal"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_marginLeft="50px"
        android:layout_marginRight="50px"
        android:layout_marginTop="50px"
        android:gravity="center"/>
  </LinearLayout>
</LinearLayout>
</ScrollView>

A.3.2 main.xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#fff">
  <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"/>
android:layout_height="wrap_content">
<EditText
android:id="@+id/paciente"
android:layout_marginLeft="15px"
android:layout_marginRight="15px"
android:layout_marginTop="15px"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
<ImageView
android:id="@+id/fbf"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:layout_marginTop="50px"
android:src="@drawable/fatebenefratelli"
/>
<Button
android:id="@+id/start"
android:text="@string/start"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textColor="#000000"
android:textStyle="normal"
android:textAppearance="?android:attr/textAppearanceMedium"
android:layout_marginLeft="50px"
android:layout_marginRight="50px"
android:layout_marginTop="50px"
android:gravity="center"
/>
<Button
android:id="@+id/about"
android:text="@string/about"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textColor="#000000"
android:textStyle="normal"
android:textAppearance="?android:attr/textAppearanceMedium"
android:layout_marginLeft="50px"
android:layout_marginRight="50px"
android:layout_marginTop="50px"
android:gravity="center"
/>
<Button
android:id="@+id/exit"
android:text="@string/exit"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textColor="#000000"
android:textStyle="normal"
android:textAppearance="?android:attr/textAppearanceMedium"
android:layout_marginLeft="50px"
android:layout_marginRight="50px"
android:gravity="center"
/>
</LinearLayout>
</LinearLayout>
A.3.3  screen1.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/texto"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/a"
        android:textSize="15sp"
        android:background="#ff000000"
        android:textColor="#fff"
        android:textStyle="bold"/>
    <Button
        android:id="@+id/SETUP"
        android:text="@string/SETUP"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:textStyle="normal"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_marginTop="80px"
        android:layout_marginLeft="50px"
        android:layout_marginRight="50px"/>
    <Button
        android:id="@+id/back"
        android:text="@string/back"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:textStyle="normal"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_marginLeft="50px"
        android:layout_marginRight="50px"/>
</LinearLayout>
```

A.3.4  screen2.xml

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"
        >
    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"
        >
```
A.4  /inPress/res/values

A.4.1  strings.xml

<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello"></string>
  <string name="app_name">FBF_InPressS</string>
  <string name="a"></string>
  <string name="gps_signal_not_found">Señal de GPS no encontrada. Compruebe el estado del GPS</string>
  <string name="gps_signal_found">Señal de GPS encontrada correctamente</string>
  <string name="search">Buscando...</string>
  <string name="search_signal_gps">Buscando señal GPS</string>
  <string name="SETUP">SETUP</string>
  <string name="stop">STOP</string>
</resources>
A.5 /inPress/res/drawable

Figure A.2 - Images used in InPresS Mobile.

A.6 Arduino code

#include <MeetAndroid.h>

MeetAndroid meetAndroid;
int onboardLed = 13;

void setup()
{
    Serial.begin(115200);
    meetAndroid.registerFunction(testEvent, 'A');
    pinMode(onboardLed, OUTPUT);
    digitalWrite(onboardLed, HIGH);
}
void loop()
{
    meetAndroid.receive(); // you need to keep this in your loop() to receive events
}

void testEvent(byte flag, byte numOfValues)
{
    flushLed(300);
    flushLed(300);
}

void flushLed(int time)
{
    digitalWrite(onboardLed, LOW);
    delay(time);
    digitalWrite(onboardLed, HIGH);
    delay(time);
}
Bibliography


