

POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA



Tesi di Laurea Magistrale

**Google Glass Data Visualization and
Monitoring for Organs-on-a-Chip and
Biomedical Applications**

Relatore

Prof. Danilo Demarchi

Candidato
Fabio Busignani s197883

Marzo 2015

ABSTRACT



The present scripture represent the master thesis of Fabio Busignani, and it has been carried out at *Khademhosseini Lab* (Cambridge, MA, USA), Harvard-MIT Health Science and Technology, Brigham and Women's Hospital under the supervision of professor Ali Khademhosseini and Ph.D. Yu Shrike Zhang.

The design which is going to be described, has been inserted inside the context of a five years project (*XCEL* grant), sponsored by the U.S. Defense Threat Reduction Agency (*DTRA*).

The aim of *XCEL* is to develop a *Body-On-A-Chip* microfluidic platform that is able to simulate multi-tissue interactions under physiological fluid flow conditions.

This master thesis will focus on designing a custom user interface on *Google Glass* for simultaneous recording of biosensing data such as temperature, pH, and microscopy images/videos as well as remote control of microfluidic valves and devices. The project involves all the hierarchical layers, starting from the physical one with the circuit in charge to acquire data from bio-sensors and drive the valves, up to the glasswear¹.

In the Introduction chapter, the main keys of the project are presented in detail as well as the final result from a user point of view.

After that, a detailed description of each abstraction level which goes to build the entire systems is shown: starting from the bottom (Hardware) reaching the top (*Google Glass Application*) passing through the Firmware, that runs in an embedded *Linux* platform, and the Software, present on the *PC* and the *Google App Engine*.

The Experiments and Conclusion chapter shows the obtained results with different experiments. The thesis ends discussing the limitations and improvements of the system.

¹ Google Glass Application

CONTENTS

Abstract	I
INTRODUCTION	1
The Glasswear	3
The Board	5
Video Storing	6
i HARDWARE	9
1 CONDITIONING CIRCUIT AND ELECTROVALVES DRIVERS	11
1.1 The Sensors	11
1.1.1 PH Sensor	11
1.1.2 Temperature Sensor	12
1.2 PH Conditioning	14
1.3 Temperature Conditioning	19
1.4 Electrovalves Driver	20
1.5 DC-DC Converter	21
1.5.1 Components Choice	22
1.5.2 Relay	24
2 PCB	26
ii FIRMWARE	31
3 INTRODUCTION	32
3.1 A Brief Introduction To IoT	33
3.2 The Beaglebone Black	34
4 EMBEDDED LINUX INSIDE THE PROJECT	36
4.1 Video Processing and Beating Plot	38
4.2 Sensor Data Acquisition	38
4.3 Electrovalves Updating Task	39
iii SOFTWARE	41
5 GOOGLE APP ENGINE	43
5.1 The web application	44
6 MICROSCOPE VIDEO STORING	47
iv GOOGLE GLASS APPLICATION	49
7 WEARABLE COMPUTING - GOOGLE GLASS	50
7.1 Google Glass	50
7.1.1 Mirror API	51
7.1.2 Glass Development Kit	51

8 THE GLASSWARE	52
8.1 The Classes	54
8.1.1 Main Service	54
8.1.2 App Drawer	54
8.1.3 Menu Activity	54
8.1.4 App Manager	56
8.1.5 Main View	56
8.1.6 PH and Temperature Views	56
8.1.7 Beating View	57
8.1.8 Electrovalves View	57
8.1.9 Video Activity	58
v CONCLUSION AND APPENDIX	59
9 TEST AND PERFORMANCE	60
9.1 LED Experiments	60
9.2 Electrovalves experiments	62
9.2.1 Breadboard Phase	62
9.2.2 PCB Phase	62
10 LIMITATIONS AND IMPROVEMENTS	63
A CODE	64
A.1 Firmware	64
A.2 Video Storing Software	75
A.3 Google App Engine	81
A.4 Glassware	88
List of Figures	144
List of Listings	145
Bibliography	147

INTRODUCTION

This master thesis has been carried out at Khademhosseini laboratory, Harvard-MIT Health Science and Technology (Brigham and Women's Hospital), in Cambridge, MA. During my six-months of research I have joined **XCEL** grant project, a five years project sponsored by the U.S. Defense Threat Reduction Agency (**DTRA**).

The goal of this project is to develop a system, a microscale bioreactor containing four 3D fully-functional *organs-on-a-chip*.

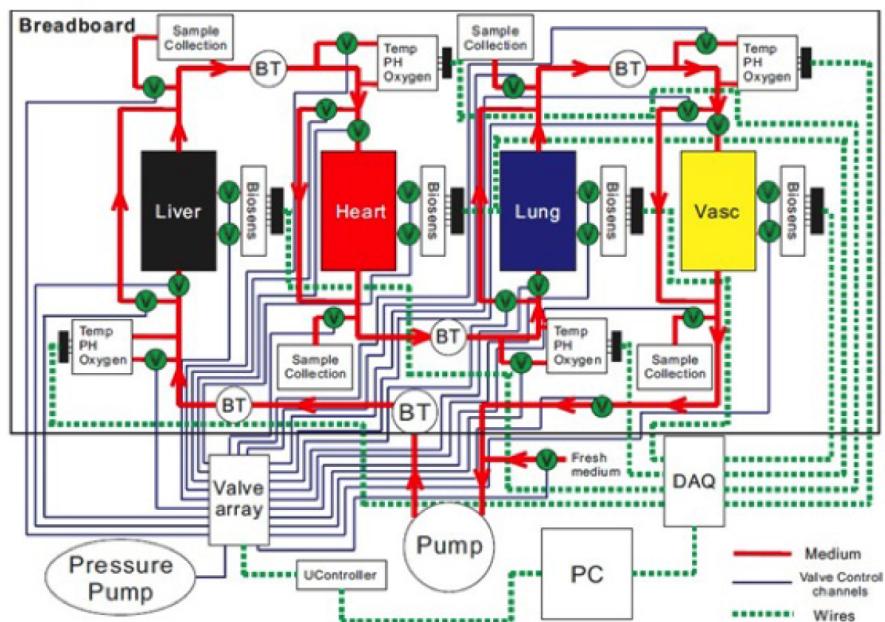


Figure 1: XCEL project (*Body-on-a-chip*)

The (Fig.1) shows, in a schematic representation, the design of the entire XCEL project. On the breadboard four organs are connected to each others: liver, heart, lung, and vascular system (*Vasc*). The medium used to connect them is a tubes circuit where the media flows. This tubes connections are driven by electrovalves.

My role in this project has been to create a custom user interface on Google Glass for simultaneous recording of biosensing data such as temperature, pH, and microscopy images/videos as well as remote control of the microfluidic valves previously introduced. In summary my aim was to design a Google Glass App to use in *organs-on-a-chip* platforms.

The *organs-on-a-chip* platforms contain interconnected microfluidic modular components including the bioreactors for hosting biomimicry human organ models, downstream biochemical sensors to continually monitor the levels of biomarkers secreted by the organs, and physical sensors to monitor the physical microenvironment of the circulatory system. Due to their extensive similarity with human organs, these miniature human models are finding widespread applications where the prediction of *in vivo* responses of

the human body is needed, including but not limited to drug screening, basic biomedical studies, and environmental safety assessment. Thus, the *organs-on-a-chip* platforms seek to recapitulate human organ function at micro-scale by integrating microfluidic networks with three-dimensional organ models, which are expected to provide robust and accurate predictions of drug/toxin effects in human bodies. In fulfilling this aim, a set of physical/chemical parameters need to be monitored and stored in order to capture such effects of drug/toxin administered into the system.

By precisely designing the Google Glass App for this organs-on-a-chip platform it allows convenient observation and control of the organ models, biosensors, and the microfluidic circuitry, which has been difficult to achieve previously.

The system designed and described in this thesis is illustrated in (Fig.2).

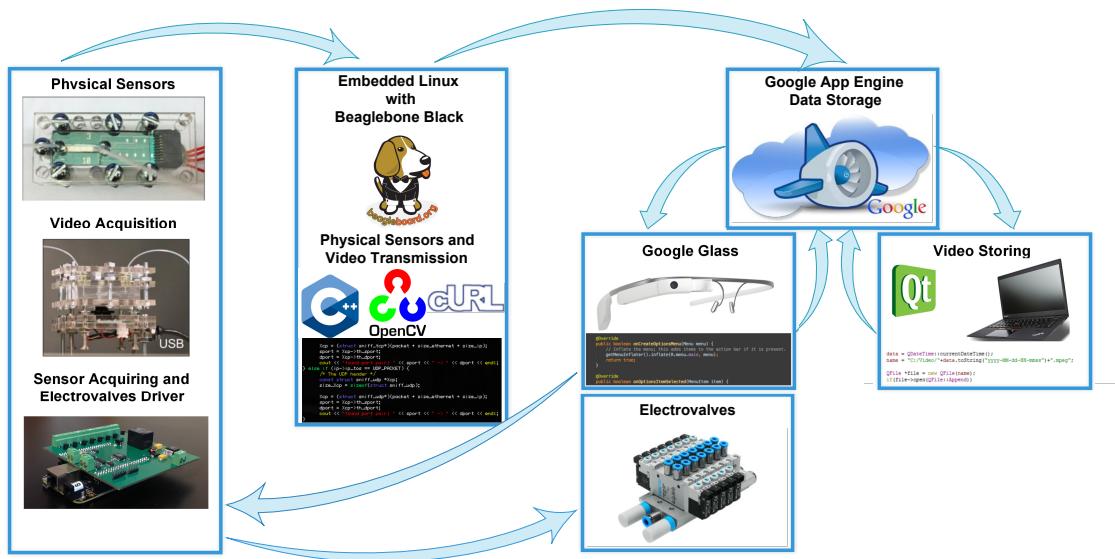


Figure 2: Block diagram of the system

The (Fig.2) shows the principal steps of data transmission from physical and video sensors to the Google Glass via an *Embedded Linux System* performed using the **Beaglebone Black**.

The Beaglebone Black runs processes that are in charged to:

- acquire the sensors value and to store them onto *Google App Engine Data Storage*;
- acquire the video, elaborate it (using *OpenCV*) in order to perform the beating plot, and to store them onto *Google App Engine Data Storage*;
- get from the *Google App Engine Data Storage* the electrovalves status set from the user through the Google Glass and to drive the valves.

The whole designed environment includes a program, written using the framework *Qt*, for storing the recorded video from microscope.

THE GLASSWEAR

The (Fig.3) shows the structure of the Glasswear. From the Home Screen (Fig.3a), using the voice trigger "*Show Measurement*" or tapping on the "*Measurement*" card (Fig.3b) user is allowed to enter the application (Fig.3c). From this point, tapping and swiping, it is possible to navigate into the glasswear's menu (Fig.3d-h) and choose which card has to be shown. *View PH* (Fig.3i) and *View Temperature* (Fig.3j) cards plot on the card's left side the value of pH and temperature, respectively. While on the right side they show the average value. The microscope's video is shown by tapping on *View Video* (Fig.3k). The *View Beating* card shows the graph of the beating associated to the video. The *View Beating* card shows the graph of the beating associated to the video.

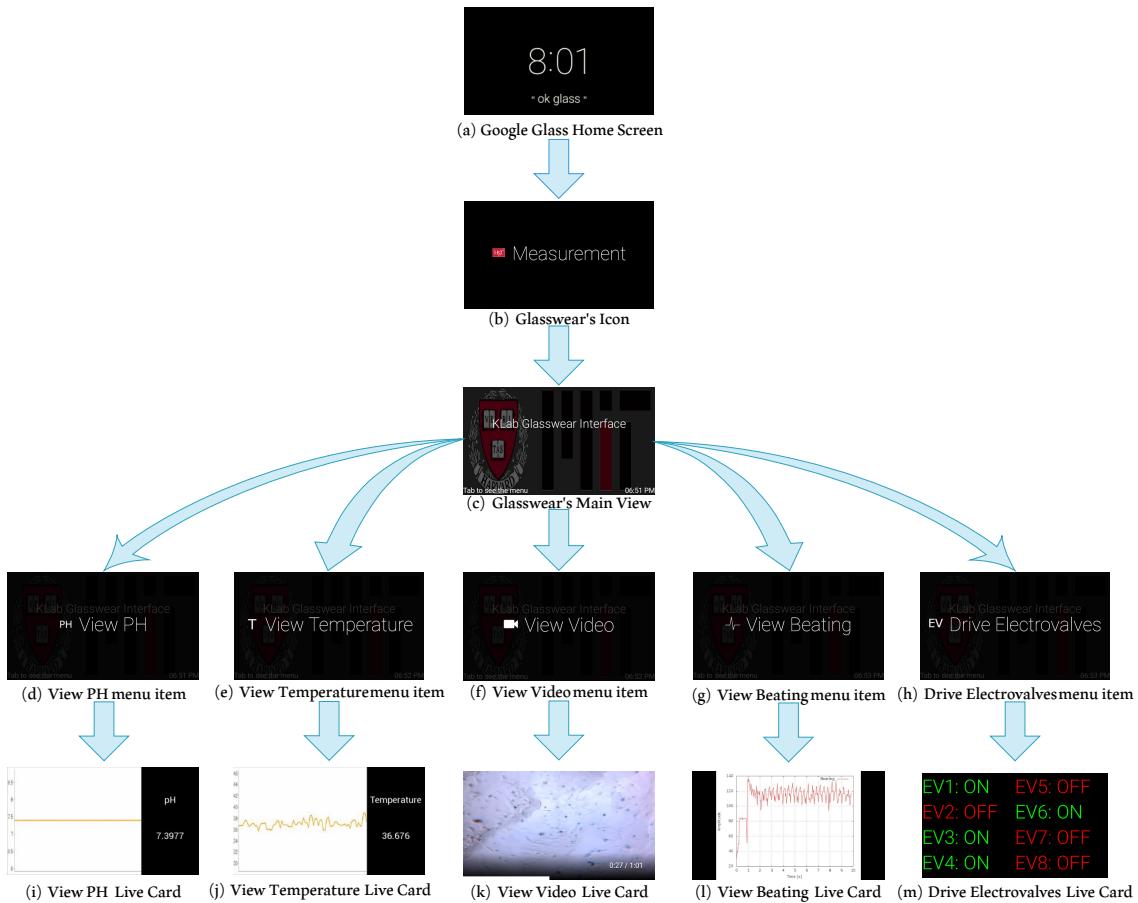


Figure 3: Glasswear's Block Diagram

From the *Drive Electrovalves* card (Fig.3m), the user can set the value of each electrovalve. The main view of this card shows the status of each electrovalve (written in green if it is on and in red if it is off).



Figure 4: Drive Electrovalves Steps

The (Fig.4) shows the steps to toggle the status of the first electrovalve:

1. (Fig.4a) shows the initial status of the whole electrovalves (all off);
2. tapping on the card and swiping the user is allowed to change the status of each electrovalve from the menu, as shown in (Fig.4b);
3. after that the electrovalve has been chosen, a toast message pops up (Fig.4c), and the new values of the electrovalves are shown.

To return on the main card of the glassware, the user has to tab on *Back* item (Fig.5) from every menu.

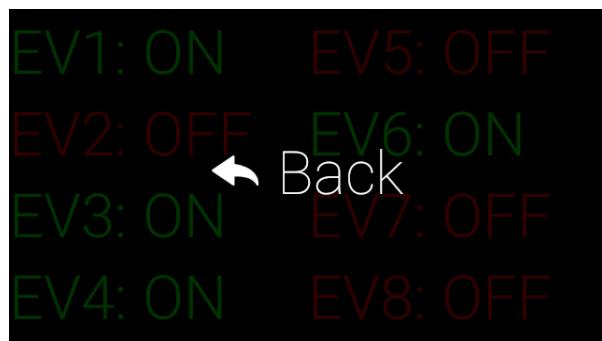


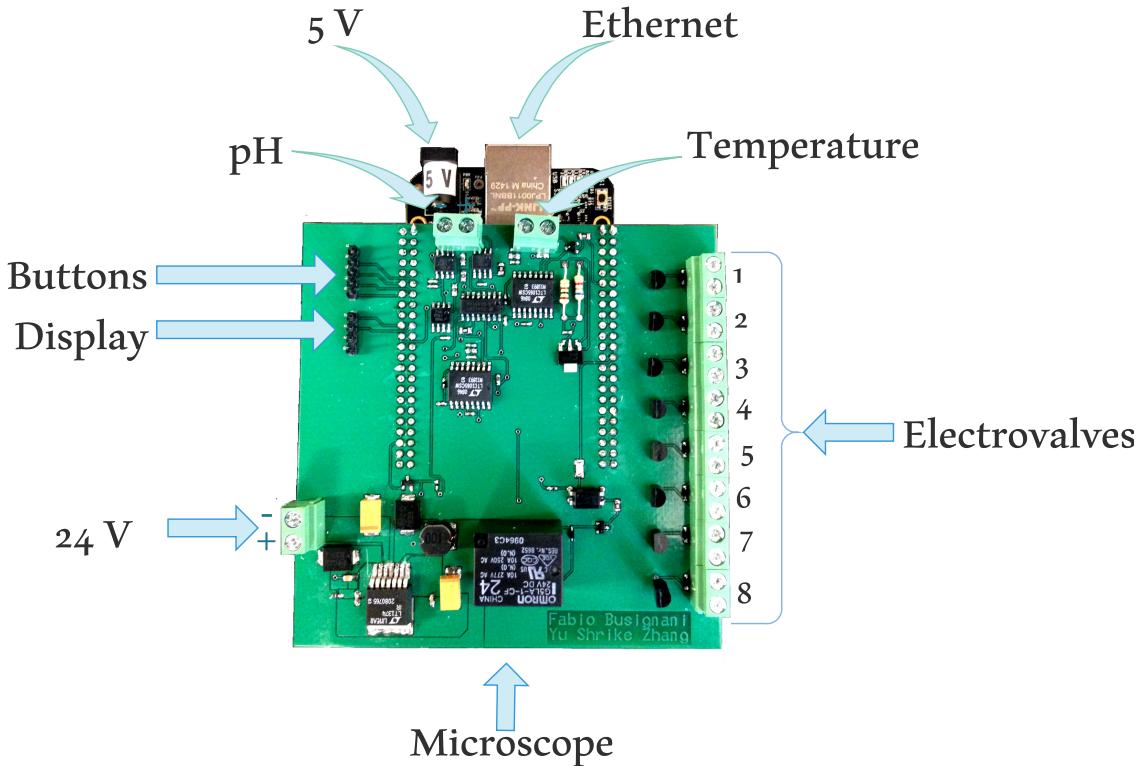
Figure 5: *Back* menu item

To terminate the glassware, from every menu, the user has to swipe up to the final item and tab on *Exit* item (Fig.6).



Figure 6: *Exit* menu item

THE BOARD

**Figure 7:** The Board

The (Fig.7) shows the top view of the system board. As can be seen it is composed by different interface and connections:

- 5 V power supply, required by the microcomputer on the Beaglebone Black and by the conditioning circuits on the *PCB*;
- 24 V power supply, required by the electrovalves;
- *Ethernet*, to connect the board to the Internet;
- *USB* connection, for the microscope;
- *pH header*, to connect the pH sensor²;
- *temperature header*, to connect the temperature sensor;
- *electrovalves header*, made by eight pairs of terminals, ordered as shown in (Fig.7), from the top to the bottom.

The remaining two headers, shown in the top left corner of (Fig.7) are thought for future application. In particular they are going to be useful for all those jobs that don't require the interaction with Google Glass, such as sensors calibration.

² **WARNING:** to ensure the correct functionality, user has to pay attention at this connection, since the pH sensor is a passive one, it has a polarity. The positive pin of the sensor has to be connected to the right terminal, looking fro the top (as shown in (Fig.7))

VIDEO STORING

The storing of the microscope video plays an important role of this system. It may be essential to review the recorded video during the experiment and in order to fulfill this aim a *Qt* program has been designed.

I chose *Qt* because in this way the program is available for different operating systems, *Linux*, *Windows*, and *MacOS*.

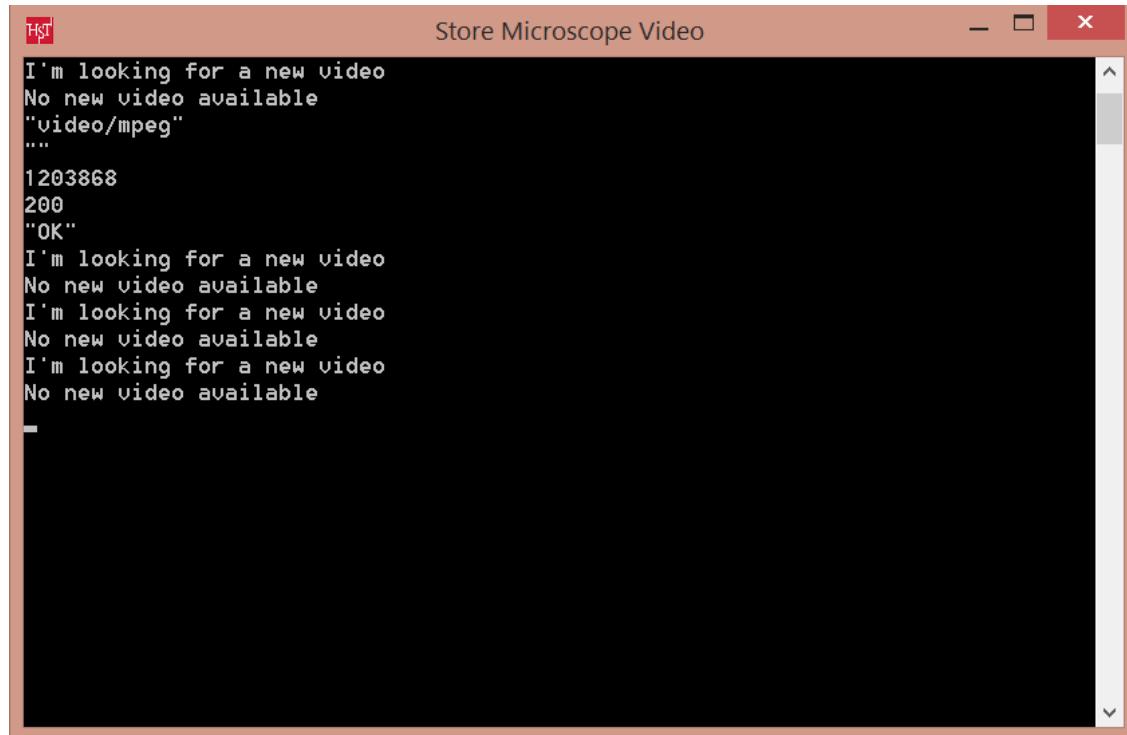
The program is very easy to use, the user just has to run the executable (Fig.8).



Figure 8: Storing Microscope Video Program's Icon

Once it has been launched, a console is opened (Fig.9). The program checks every 20 seconds if a new video has been uploaded on the server. If so, the new video will be stored inside the computer (directory C:/Video) with the current data and hour as name in the following form: *YYYY-MM-DD-HH-mmss*, as shown in (Fig.10).

As shown in (Fig.9) on the console the user can read all the information about what the program is doing.



```
I'm looking for a new video
No new video available
"video/mpeg"
...
1203868
200
"OK"
I'm looking for a new video
No new video available
I'm looking for a new video
No new video available
I'm looking for a new video
No new video available
```

Figure 9: Storing Microscope Video Console

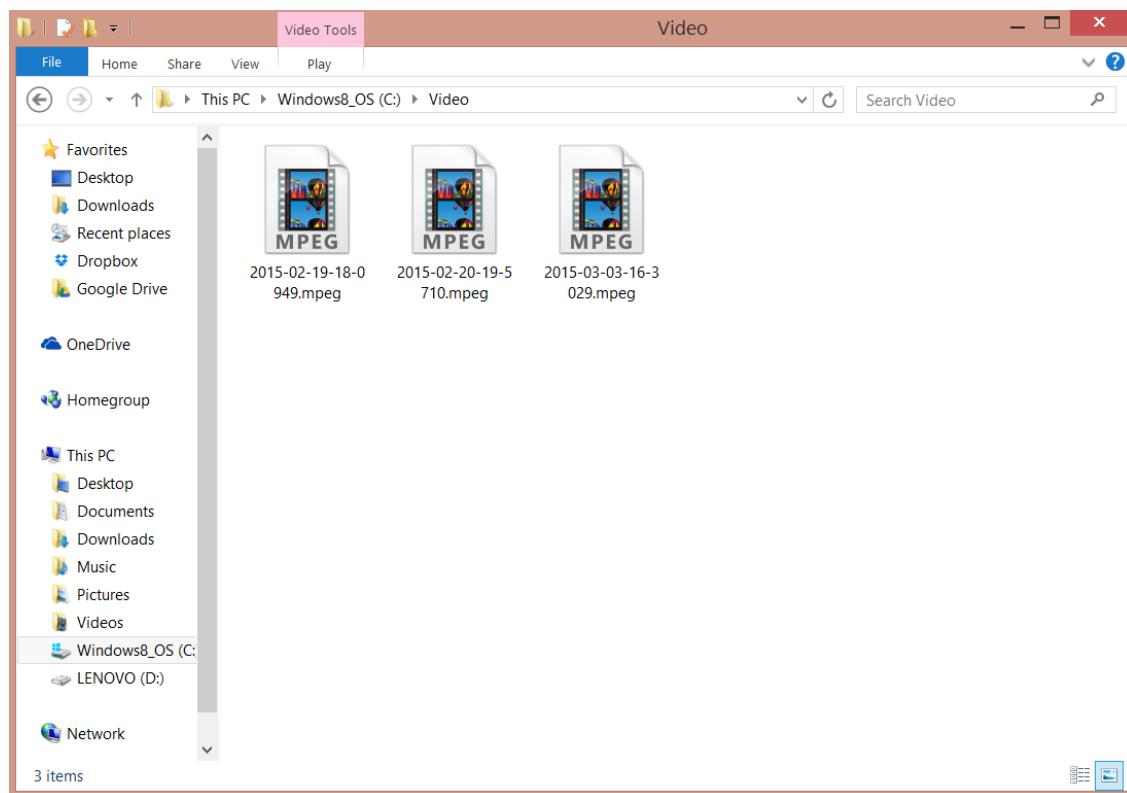


Figure 10: Video Stored in the Folder

Part I

HARDWARE

In this part of the thesis the hardware that has been designed in this system is explained. The design specification for this part are:

- make a sensor conditioning for pH and temperature sensors, see (Sec.1.2) and (Sec.1.3) for more details;
- make a driver for electrovalves which must be able to drives two different kinds of electrovalves, both of them require 80 mA but one type at 24 V while the other one at 12 V . To fulfill this aim a *DC-DC* converter has been designed, as well. See (Sec.1.4) for more details;
- make a *PCB* which supports and connects all the previous components and that is a *capes* for the Beaglebone Black, it has to be wedged on top of it, see (Chap.2) for more details.

1

CONDITIONING CIRCUIT AND ELECTROVALVES DRIVERS

1.1 THE SENSORS

The sensors used in this project are microfabricated on a single silicon chip, and kindly provided by professor Dr. Sandro Carrara research group from École Polytechnique Fédérale de Lausanne (*EPFL*), Switzerland.

The choice of using a microfabricated sensor instead of a commercial one (the number of commercial sensors is very high even for biomedical applications) has been taken because of a remarkably decrease in sensing occupied space. Indeed, in this way, the area consumption can be optimized.

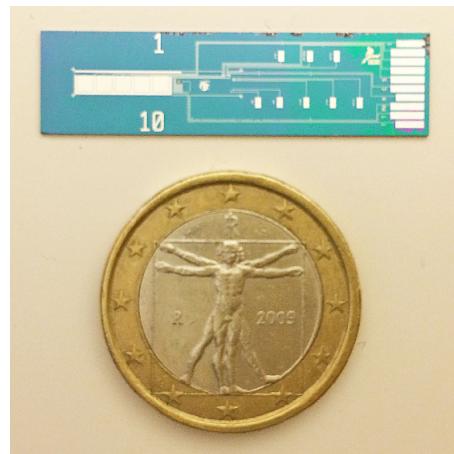


Figure 11: Multisensor - Size Comparison

The (Fig.11) shows the sensor and highlights its very small dimension ($35.1 \times 9.3 \text{ mm}$). This multisensor contains an Iridium Oxide based pH sensor and a platinum resistance temperature detector (*RTD*) [1]. The device also hosts five independent working electrodes (*WE*), with shared platinum reference electrode (*RE*) and platinum counter electrode (*CE*). But, at least for the time being, we are not going to use them.

The (Fig.12) shows the circuit schematic of multisensor. As can be seen, from the interface the pads that are going to be used in this thesis are the four on the bottom. In fact, the last pair of pins are the temperature ones, and the penultimate ones are tied to the pH sensor (negative pin on top).

1.1.1 PH Sensor

The embedded pH sensor has been made by platinum. Its electrical output is a voltage value which varies almost linearly with the pH.

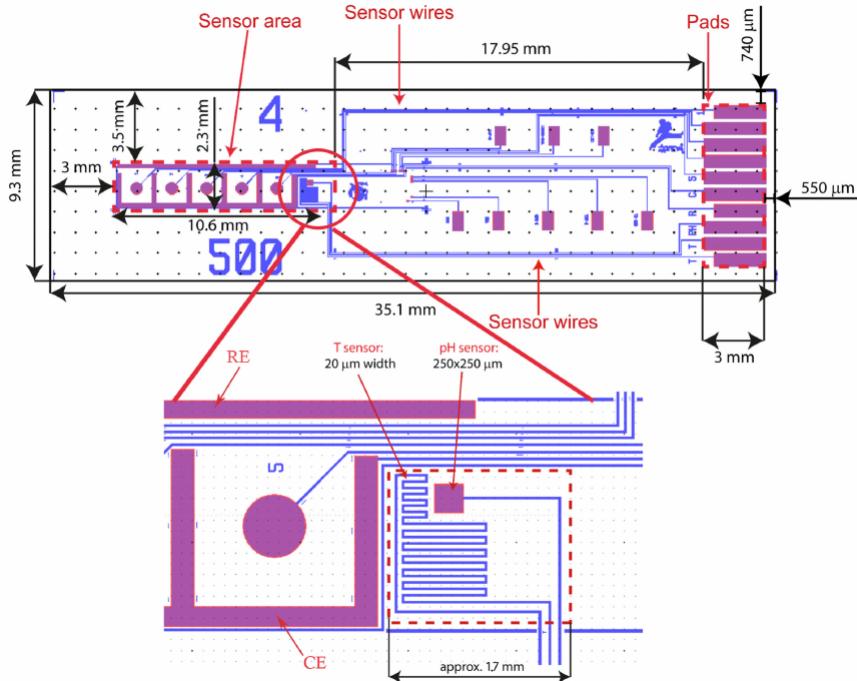


Figure 12: Multisensor - Scheme

The pH sensor needs to be calibrated before it can be used. The calibration and characterization of sensor are made in different environments and with different solutions. After this step the transfer function of the sensor is linear (Fig.14) with a slope of around -100 mV/pH .

1.1.2 Temperature Sensor

For almost all the metal materials, electrical resistance is a parameter which varies with temperature. In order to create a resistance temperature detector (*RTD*), a coupled system of a metal wire and a resistance measurement device is needed. Inside the used device, the temperature sensor is made by a platinum wire. This metal is strongly used for temperature applications because it allows to obtain the most accurate measurements (up to $\pm 0.001 \text{ }^{\circ}\text{C}$) with a linear output response.

The relationship between resistance and temperature of a platinum resistance thermometers is described by the ***Callendar-Van Dusen*** equation (Eq.1).

$$R(T) = R(0) \cdot [1 + A \cdot T + B \cdot T^2 + (T - 100) \cdot C \cdot T^3] \quad (1)$$

Where:

- $R(T)$, is the resistance at the T temperature;
- $R(0)$, is the resistance value at $0 \text{ }^{\circ}\text{C}$;

- A , B , and C , are the *Callendar-Van Dusen constants* defined by the following:

$$A = \alpha + \frac{\alpha \cdot \delta}{100}, \quad (2a)$$

$$B = -\frac{\alpha \cdot \delta}{100^2}, \quad (2b)$$

$$C = -\frac{\alpha \cdot \beta}{100^4}. \quad (2c)$$

And $\alpha = 0.003921 \Omega/\text{ }^\circ\text{C}$, $\beta = 0$ for positive temperature and $\delta = 1.49$.

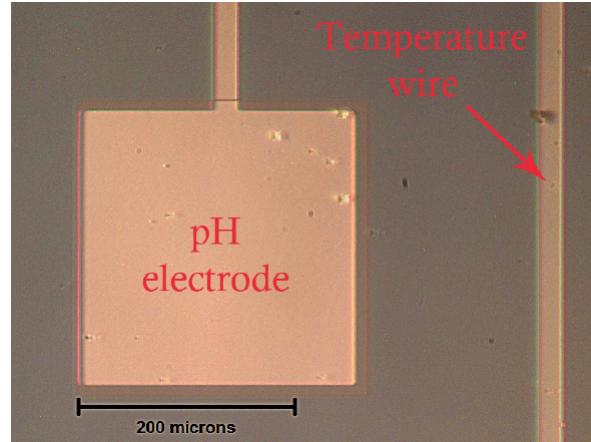


Figure 13: PH and Temperature Sensors from an Optical Image

The pH and temperature sensors are very close from each others (Fig.13), and this is very important because the pH electrode's sensitivity varies over temperature, thus: temperature affects the pH value, as can be seen from (Eq.3). So, we always need to know at what temperature we make the pH measure.

$$\text{pH}(X) = \text{pH}(S) + \frac{(E_S - E_X) \cdot F}{R \cdot T \cdot \ln(10)} \quad (3)$$

The (Eq.3) represents the transfer function of a pH sensor, where:

- $\text{pH}(X)$, is the pH value of unknown solution;
- $\text{pH}(S)$, is the pH value of standard solution (7);
- E_S , is the electric potential at standard electrode;
- E_X , is the electric potential at pH-measuring electrode;
- F , is the Faraday constant ($9.6485309 \cdot 10^4 \text{ C mol}^{-1}$);
- R , is the universal gas constant ($8.314510 \text{ J K}^{-1} \text{ mol}^{-1}$);
- T , is the temperature in Kelvin.

1.2 PH CONDITIONING

As already explained in (Sec.1.1.1), the pH sensor is a passive sensor, which means no excitation source is required because the sensor itself generates its own electrical output signal. In particular any variation of pH in input is transduced in a voltage variation in output.

The pH sensor is also bipolar, this means the voltage output may be both positive and negative, as shown in (Fig.14).

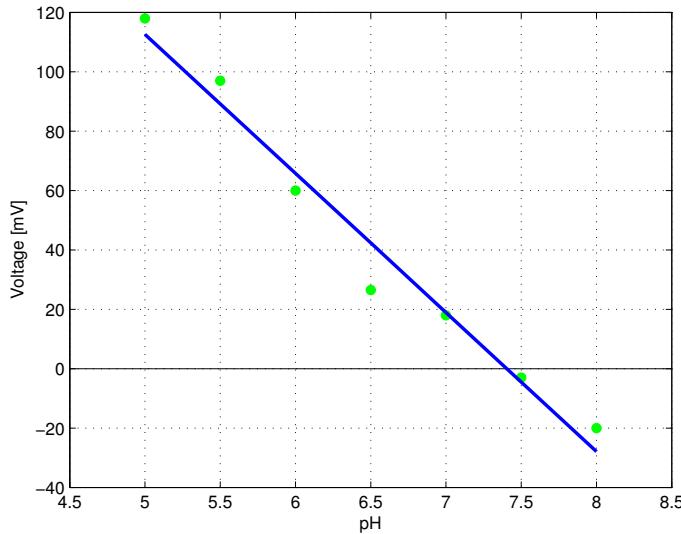


Figure 14: Typical pH-sensor transfer function

Resuming, it produce a voltage output that decreases linearly with pH of the solution being measured. The sensors give a sensitivity which ranges between 50 and 120 mV/pH (it depends from sensor to sensor), this means that, in order to well observe this variation, an amplification stage may be required.

The (Fig.15) shows the adopted solution for conditioning the pH sensor. First of all, since the pH sensor produces a bipolar signal and this application operates on a single voltage supply, the signal has been level shifted. To achieve this first challenge the operation amplifier $U1$ forces an off-set of 512 mV to the pH sensor. Indeed, the *LM4140A-1.0* is a high precision low noise *LDO* (Low Drop Out) voltage reference which provides an accurate 1.024 V . This voltage has been halved by the $10\text{ K}\Omega$ resistor divider. The $U1$ is in voltage follower configuration, thus its output should be equal to the input, and it biases the reference electrode of the pH sensor with 512 mV , at low impedance. So, what the part of circuit made by $U1$ and *LM4140A-1.0* does is to shift the bipolar pH sensor signal to an unipolar in order to be usable in the single-supply system.

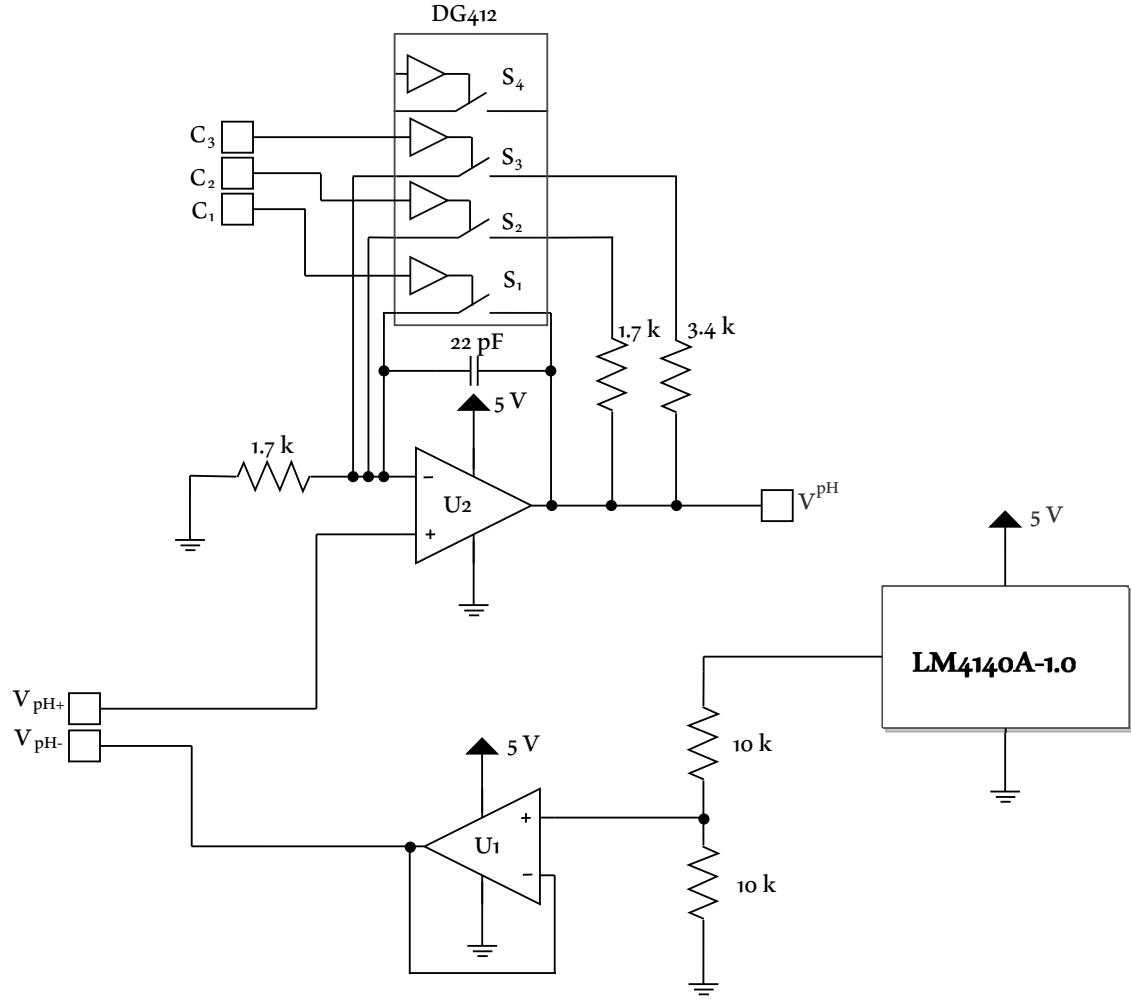


Figure 15: Conditioning circuit for pH sensor

Another challenge is given by the high impedance of the electrode. In fact the output impedance of the pH sensor is higher than $100 M\Omega$. The circuit in (Fig.16) shows a typical connection of this sensor where the output voltage is given by:

$$V_{out} \simeq V_{in} = V_S - I_{bias} \cdot R_S \quad (4)$$

Thus, in order to reduce the error caused due to amplifier's input bias current a really low input bias current amplifier has to be chosen. For this reason, the *LMP7721* is used, it is has an ultra-low input bias current ($3 \pm 17 fA$).

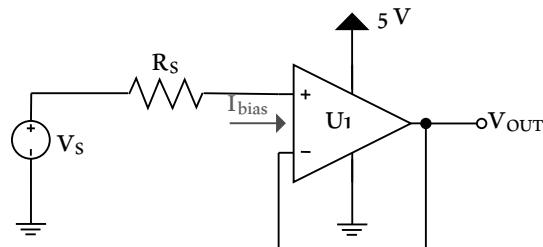


Figure 16: Error caused by Amplifier's Input Bias Current

In (Fig.15) both $U1$ an $U2$ are *LMP7721*. The second amplifier with the *DG412* represents a simple **PGA** (*Programmable Gain Amplifier*), where the resistors have been chosen to give the following gains:

- 1, when C_1 is asserted and the others are denied;
- 2, when C_2 is asserted and the others are denied;
- 4, when C_3 is asserted and the others are denied.

The feedback capacitor is used to ensure stability and holds the output voltage during the switching times. Indeed, in these slice of time the output node would be floated without the capacitor.

This PGA stage introduces an additional offset error of $75 \pm 470 \text{ fV}$, due to the bias current of the operational amplifier and R_{ON} of *DG412* (25Ω). That voltage combined with the *LMP7721* offset (which is very higher than the first one) is approximately equal to $26 \mu\text{V}$.

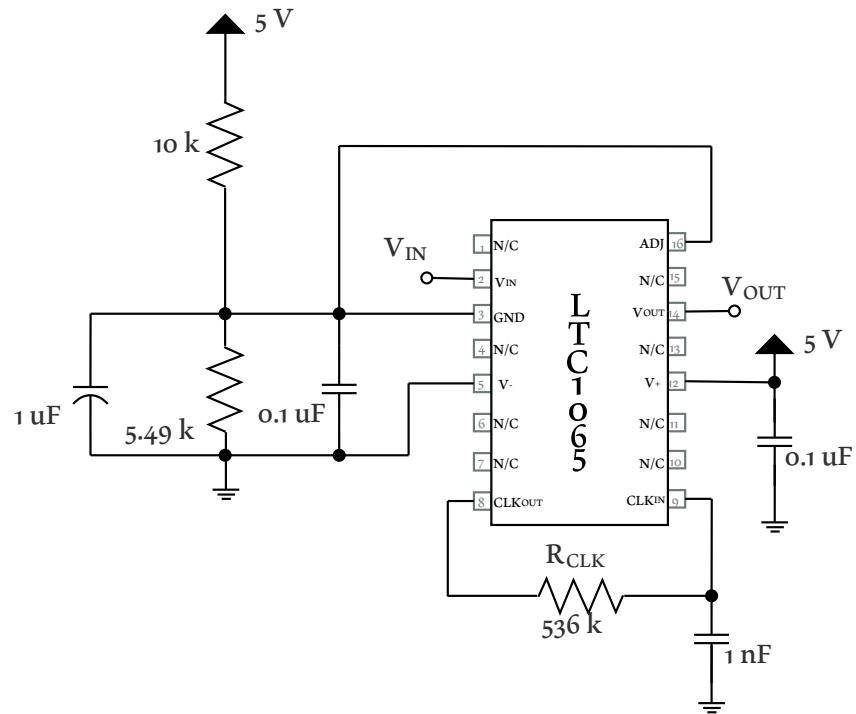
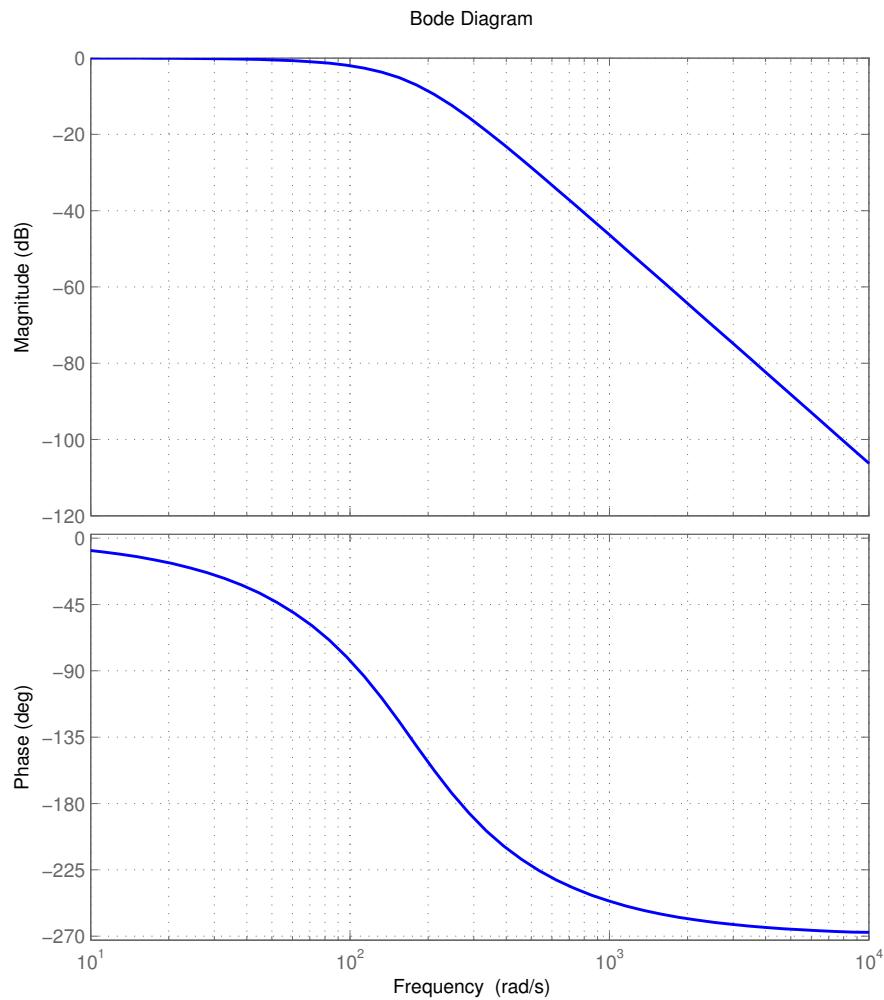
Since the environment in which the circuit is going to be used is a laboratory, so a really noisy place, the conditioning circuit for the sensor has to involve the design of a **low-pass filter** in order to reject the noise.

The circuit shown in (Fig.17) is a third order low-pass filter with a *Bessel* response. It has been designed in order to ensure a really flat-response in the pass band. The parameter of the filter are:

1. *cutoff frequency* (f_c): 26.8 Hz ;
2. *stop band attenuation*: -28.2 dB at *stop band frequency* (f_s) 60 Hz (the line frequency in USA);
3. *Quality factor* (Q): 0.65 ;
4. *filter order*: third;
5. *filter response*: Bessel.

It's important that $Q < 0.707$ because otherwise would be some peaking in the filter response. While, in this case, as shown in (Fig.18), roll-off at the cutoff frequency is greater.

This filter also behaves as *anti-aliasing filter*, to prevent the aliasing components from being sampled during the analog to digital conversion.

**Figure 17:** Low-Pass Filter Schematic**Figure 18:** Low-Pass Filter Frequency Response

The output of this filter represent the input signal of the embedded ADC inside the Beaglebone Black.

Connecting together all this part we obtain the circuit in (Fig.19) where we can also see the general purpose input/output pin used to drive the PGA and the analog input used for the pH sensor. As it is explained in (Chap.4) AN_2 is used to let the microcontroller know about the added offset.

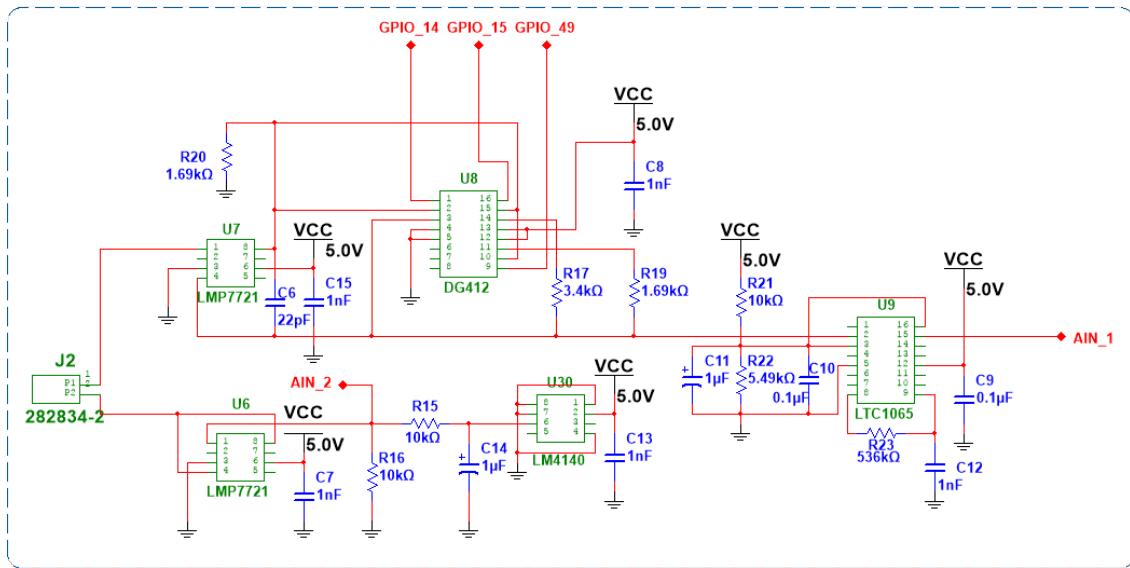


Figure 19: Acquisition Path for pH Sensor

1.3 TEMPERATURE CONDITIONING

As already explained in (Sec.1.1.2), the temperature sensor is a active sensor, which means excitation source is required because the sensor is resistor based so a current must be passed through it. Then, the corresponding voltage has to be measured in order to determine the temperature value.

So, any variation of temperature in input is transduced in a resistance variation in output.

The (Fig.20) shows the adopted solution for conditioning the temperature sensor. In this connection the temperature sensor is excited by $680 \mu A$, so the voltage V_T is given by the following equation:

$$V_T = 680\mu \cdot R_T \quad (5)$$

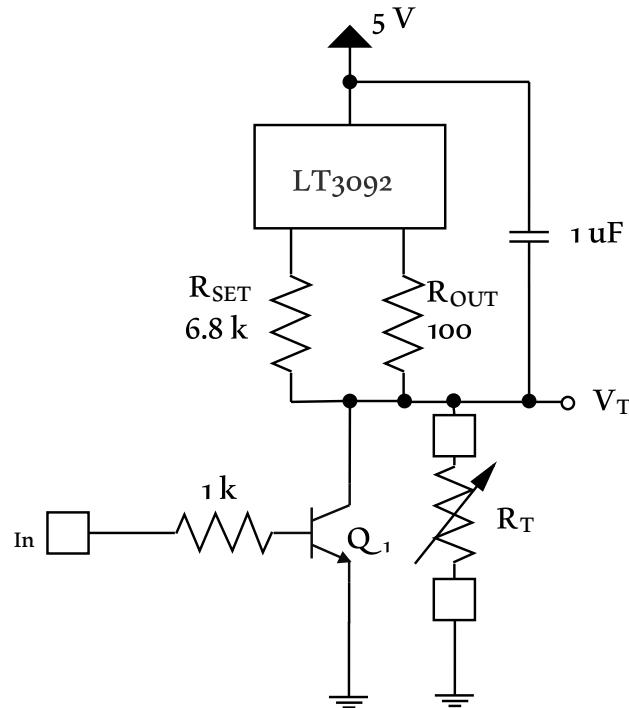


Figure 20: Conditioning Circuit for Temperature Sensor

In order to provide this amount of current a *LT3092* is used. It can supply an output current equal to:

$$I_{OUT} = 10\mu \cdot \frac{R_{SET}}{R_{OUT}} \quad (6)$$

To ensure the stability of the component, a feedback capacitor of $1 \mu F$ is exploited. The transistor Q_1 is used to avoid the self-heating of the temperature sensor: when the temperature value has to be sampled In is denied, for the remaining time In is asserted, in this way the resistive sensor is by-passed, and the Joule effect is avoided.

For the same reason exposed in (Sec.1.2), also in this case a filter is needed, and, since the voltage value is almost the same, the used filter is equivalent to the previous one.

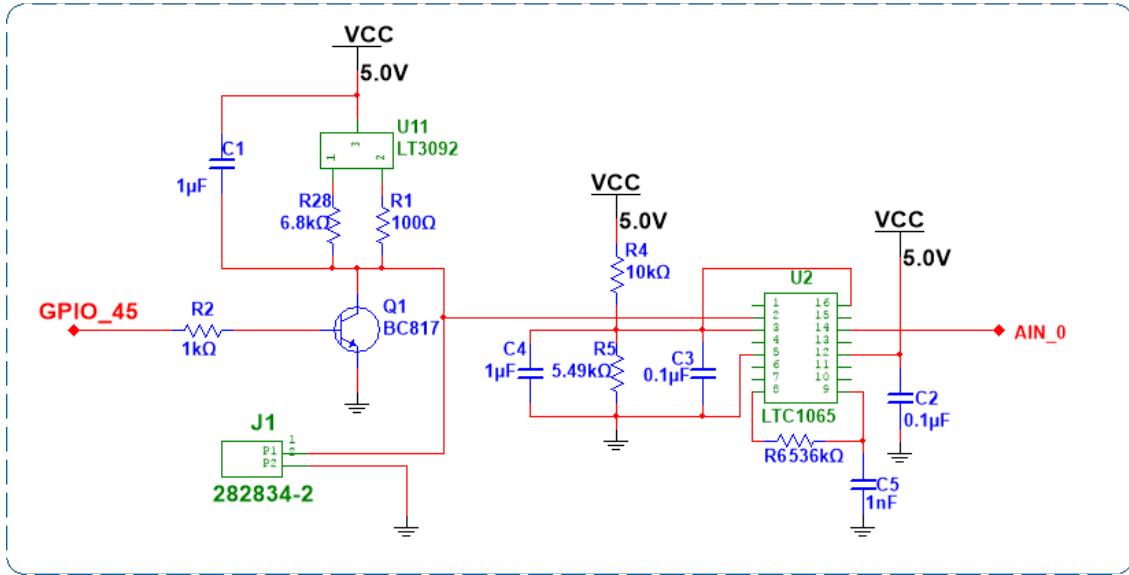


Figure 21: Acquisition Path for pH Sensor

1.4 ELECTROVALVES DRIVER

In order to allow the reverse control, from Google Glass to electrovalves, it is important to design a circuit that has to drive them from digital values provided by the *Beaglebone Black* (0 equal to 0 V, 1 equal to 3.3 V and in both cases the maximum suppliable current is only 6 mA).

The electrovalves used are FESTO solenoid valves MH1. They require a voltage of 24 V in DC and a current of 80 mA.

To fulfill this aim a low-side switch MOS has been used, as shown in (Fig.23).

The *Fairchild Semiconductor BS170 N-Channel MOS* has been chosen because of its low price and its capability of supporting a drain-source voltage of up to 60 V and supplying a continuous drain current of up to 500 mA.

To protect the *BS170* from reverse inductive current surges due to the solenoid of the electrovalve, a *Vishay Semiconductors IN4148 Diode* is used. It is able to support a reverse voltage of up to 75 V and a continuous forward current of up to 150 mA.

As shown in (Fig.22) the number of electrovalves used is eight, so the previous driver has been replicated in order to obtain the circuit in (Fig.)



Figure 22: Electrovalves

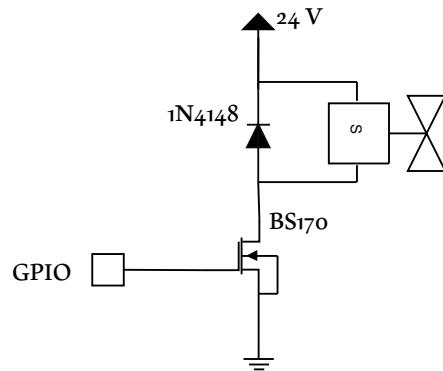


Figure 23: Driver for electrovalves

1.5 DC-DC CONVERTER

Since this system is supposed to drive different kind of electrovalves, which require a different value of voltage (but not in current) a *DC-DC converter* is used in order to convert the voltage supply from 24 V to 12 V.

Using the *LT1374* the circuit in (Fig.25) has been designed, it is a constant frequency (equal to 500 Hz), current mode buck converter. It has an embedded clock and two feedback loops to control the duty cycle of power switch.

Through a low-side switch, made with the *BS170*, it is possible to shutdown the converter from the software. When the *LT1374* is in shutdown, the supply current is reduced to $20 \mu A$. As it is explained in (Chap.4) this is used to prevent regulator from operating when 24 V are required.

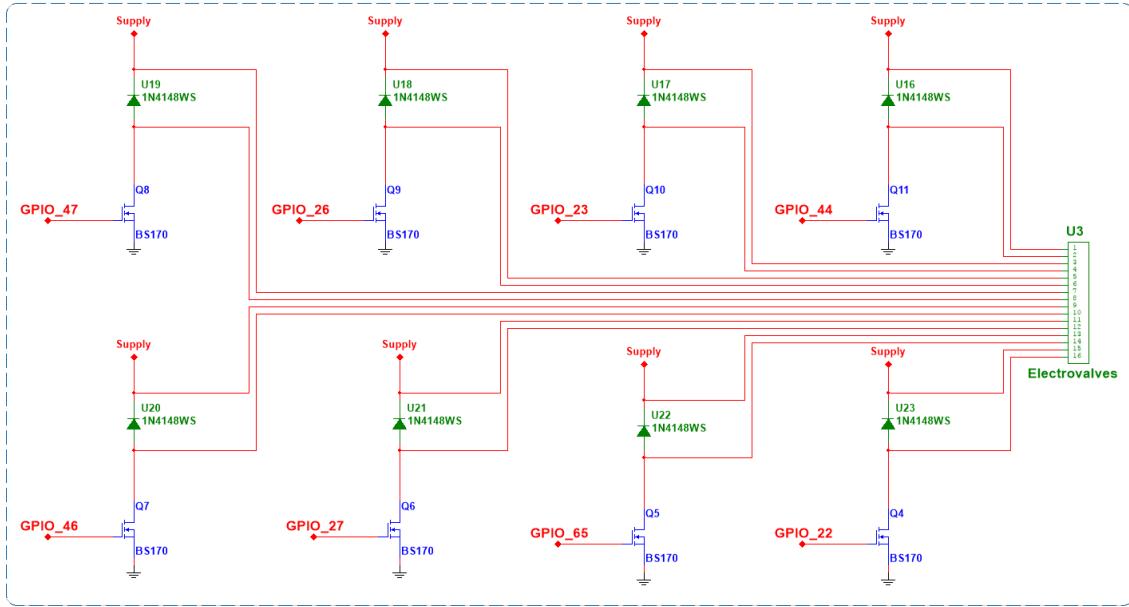


Figure 24: Electrovalves Driver Circuit

1.5.1 Components Choice

All the components have been chosen with attention and for some reasons that are going to be explained belong.

FEEDBACK RESISTORS

The main behavior of the feedback pin on the *LT1374* is to set the output voltage, and this deals with selecting the resistors R_1 and R_2 . They are related from each others by the following equation:

$$R_1 = \frac{R_2 \cdot (V_{OUT} - 2.42)}{2.42} \quad (7)$$

As suggested on datasheet of the component, the resistor between feedback pin and ground is $4.99\text{ k}\Omega$. So, from the (Eq.7) results that the value of R_1 is $19.6\text{ k}\Omega$.

INDUCTOR

The choice of inductor is a trade-off among:

- *physical area*, lower values of inductor mean lower size;
- *output current*, higher values of inductor allow more output current because they reduce peak current ($I_{SW(Peak)} \propto 1/L$)
- *ripple voltage*, higher values of inductor reduce the output ripple voltage.

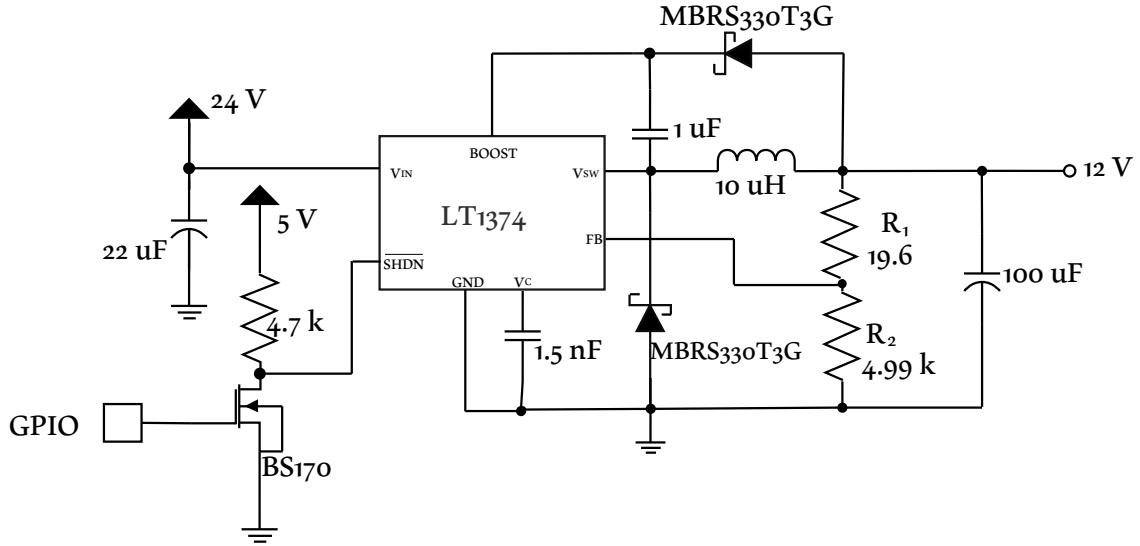


Figure 25: DC-DC circuit

A good choice is represented by $10 \mu H$, with this inductor the maximum current peak (Eq.8) is equal to $1.24 A$.

$$I_{SW(Peak)} = I_{OUT} \frac{V_{OUT} \cdot (V_{IN} - V_{OUT})}{2 \cdot f \cdot L \cdot V_{IN}} \quad (8)$$

OUTPUT CAPACITOR

The output capacitor determines the output ripple voltage, for this reason a small *Effective Series Resistance* (ESR) is required.

The frequency operation of *LT1374*, as already said, is equal to $500 Hz$ and at this frequency any polarized capacitor is essentially resistive. As suggested from datasheet, for typical *LT1374* application the ESR has to range from 0.05Ω to 0.2Ω , for this reason the output capacitor is a *solid tantalum capacitor*. The choice of $100 \mu F$ is a good trade-off between output ripple voltage and physical area.

SCHOTTKY DIODE

The chosen diode is *On Semiconductor MBR330* because of its capability of supporting a $3 A$ average forward current and $30 V$ reverse voltage.

Indeed, the reverse voltage is approximately $12 V$ (the output voltage), while the average forward current is given by the (Eq.9).

$$I_{D(Avg)} = \frac{I_{OUT} \cdot (V_{IN} - V_{OUT})}{V_{IN}} \quad (9)$$

The (Eq.9) will never yield values higher than $3 A$, neither in worst-case scenario.

BOOST CAPACITOR

The boost capacitor has been chosen based on the voltage that has to support, which is basically equal to the output voltage (12 V) and the (Eq.10) provided by *LT* on datasheet of the component. In this application result that its minimum value is equal to 1.5 nF .

$$C_{MIN} = \frac{(I_{OUT}/50) \cdot (V_{OUT}/V_{IN})}{f \cdot (V_{OUT} - 3)} \quad (10)$$

1.5.2 Relay

Since the electrovalves may need 12 V or 24 V a way to switch between this two voltages supplies is needed. The circuit shown in (Fig.26) has been used for this purpose.

It allows the switching trough the firmware. The circuit uses a optocoupler, the DPC-817C, to isolate the Beaglebone Black to the relay, and prevent in this way that pounces produced by magnetic part of relay reach the general purpose pin of the Beaglebone itself. The relay used is the G5LA1CF24DC and, as happened in (Sec.1.4), a diode has been exploited to preserve the transistor used as voltage controlled switch from broking.

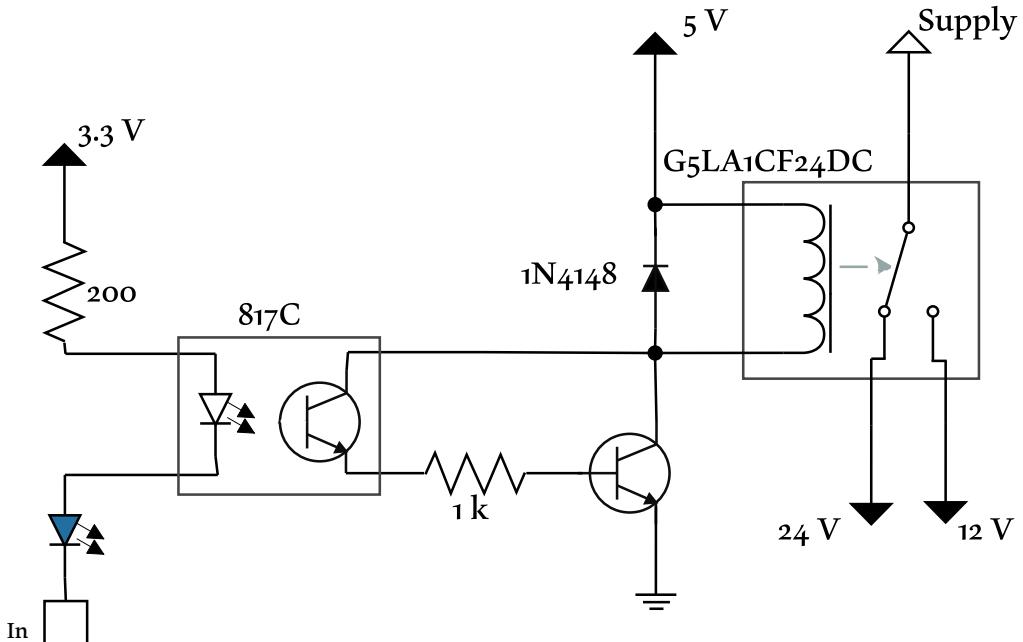


Figure 26: Relay circuit

Summary, in the circuit represented in (Fig.26) the *Supply* is the voltage which is going to be provided to the electrovalves. When the *In* signal is asserted, the relay is in its normal connection, son *Supply* is tied to 24 V . On the other hand, when *In* is denied relay is excited and *Supply* is tied to 12 V .

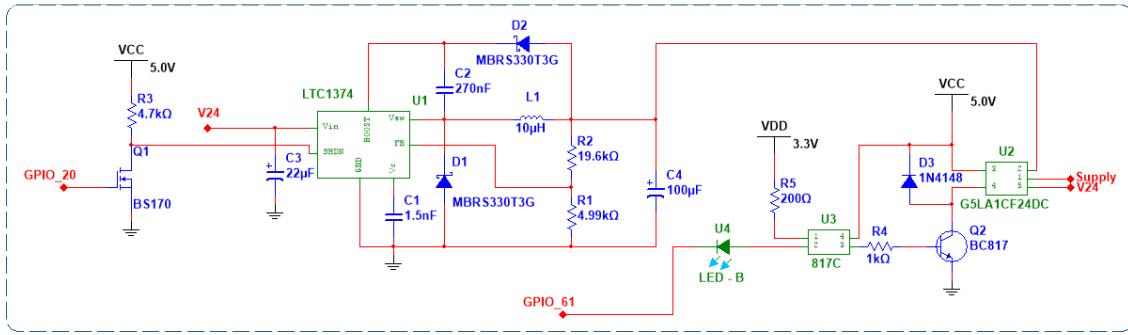


Figure 27: Electrovalves Supply Circuit

The circuit in (Fig.27) shows the connection between *DC-DC* circuit and the switching one, all of this is necessary to correctly supply the different kind of electrovalves.

2 | PCB

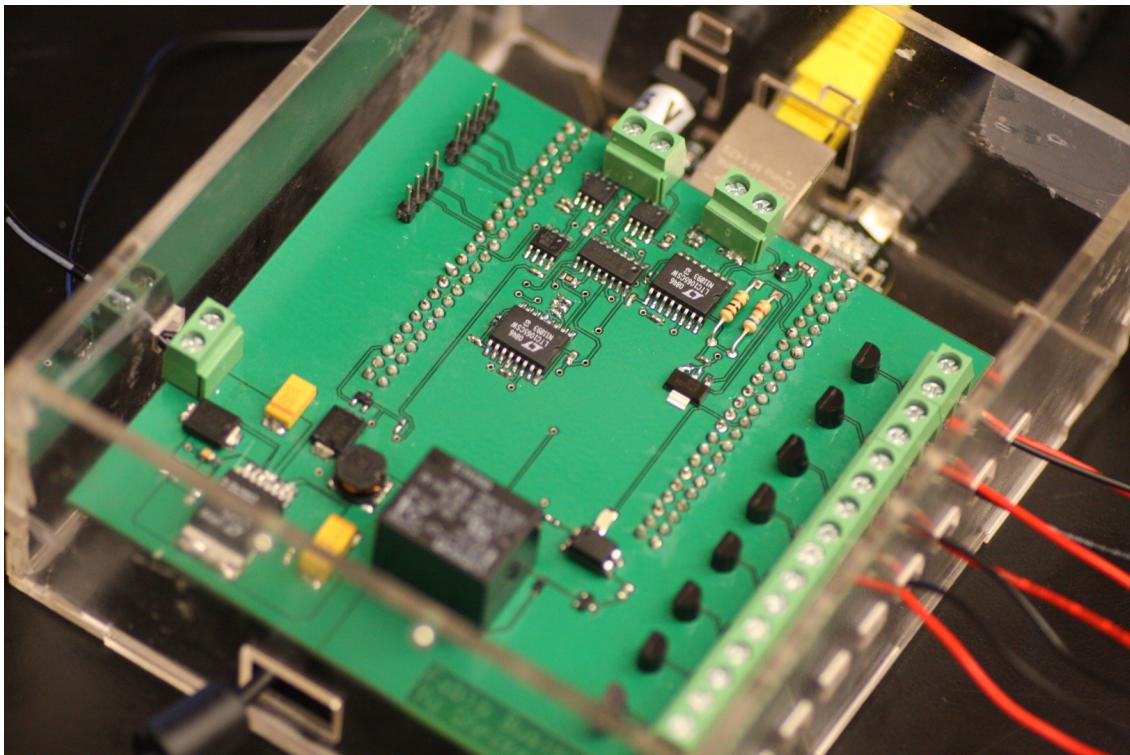


Figure 28: Final system mounted on a PCB

All the hardware and the circuit described so far has been mounted on a Printed Circuit Board (*PCB*) that is supposed to be a *cape* of the Beaglebone Black. This means that the whole physical structure has been made to be attached on the headers of the board. In (Fig.29) all the circuit that has to be made on the *PCB* is shown.

The PCB has been designed in a two-layer board ($100 \times 100 \text{ mm}$) using the *National Instruments Circuit Design Suite*, in particular *Multisim* to carry out the schematic and *Ultiboard* for what concern the PCB.

The design has followed the guidelines for reduced electromagnetic interface (*EMI*). First of all, since Surface-mount devices (*SMD*) are better than Through-hole components (*THD*) in dealing with RF energy, because of reduced inductances and closer component placements available ([2]), where there was a choice, *SMD* components have been used.

Particular attention has been paid for the *DC-DC* converter layout, because a wrong *PCB* design for switching power supply often means failure. Moreover, in these terms, what is good for *EMI* is also good in terms of functional stability for the regulator.

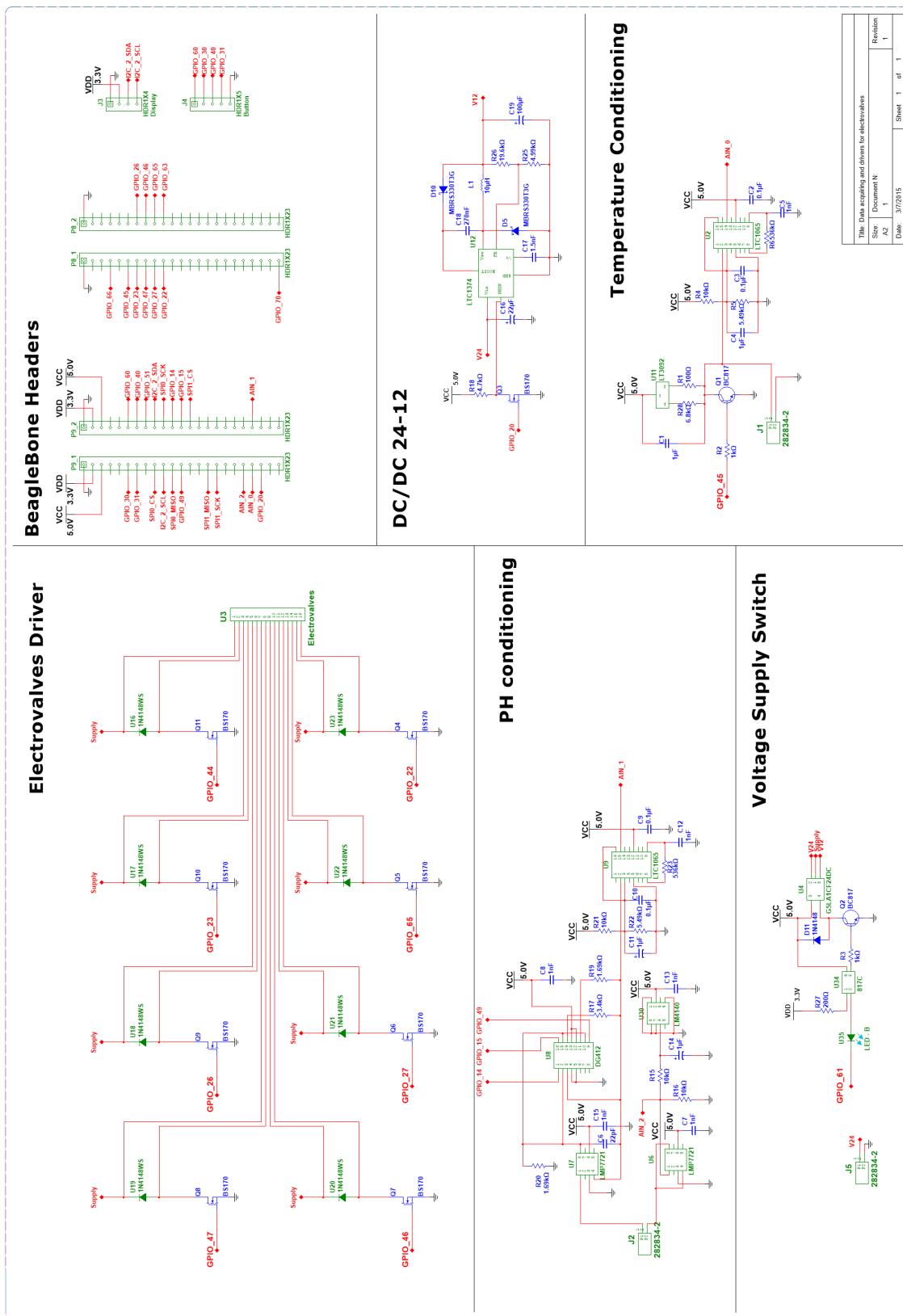


Figure 29: Schematic of Complete Circuit

The circuit in (Fig.30) is schematically a common buck regulator. In that figure, with a blue circle, is highlighted the high speed switching current path: the loop which produces the highest *EMI*. Indeed, in the blue loop flows a fully switched alternated current, and for this reason it is also referred as **hot loop**.

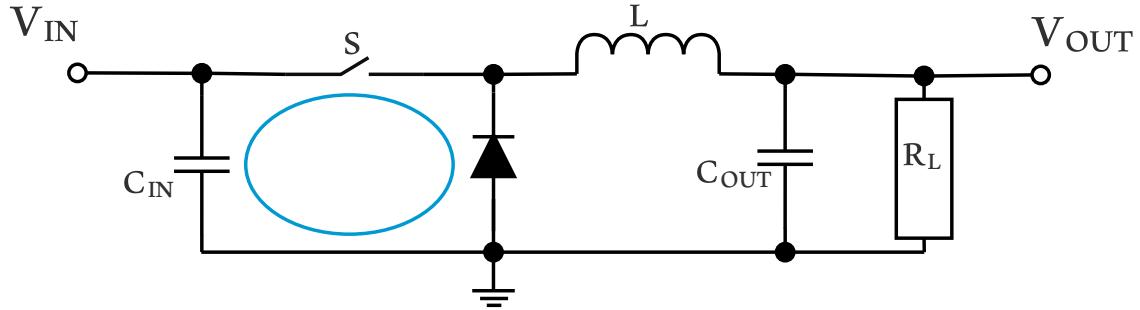


Figure 30: DC-DC High Speed Switching Path

In order to ensure clean switching and reduce *EMI* the minimum lead length is required for reducing the radiating effect of the hot loop as much as possible: and so it has been done, as can be seen from (Fig.31).

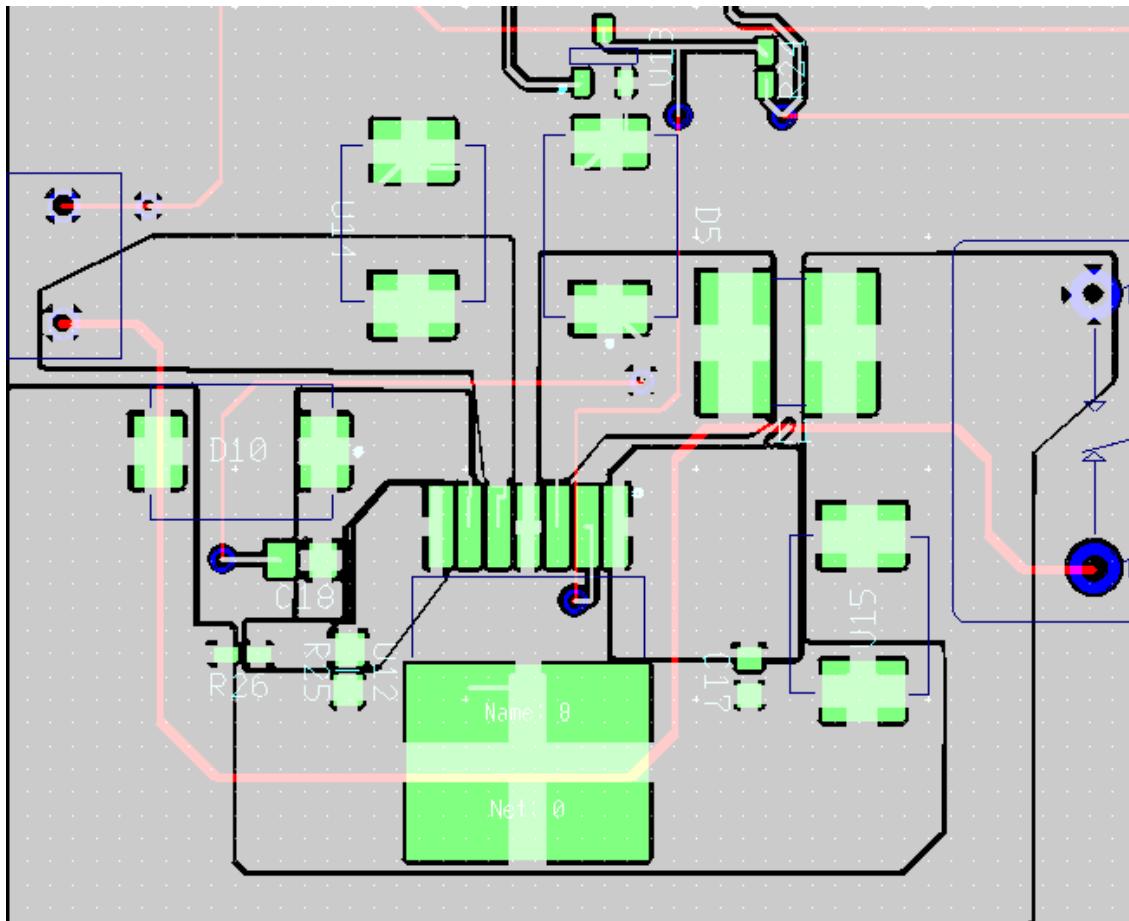


Figure 31: PCB Layout of DC-DC

In (Fig.31) the *PCB* layout for *DC-DC* converter has been shown, in which we can see that:

- the magnetic radiation is minimized by keeping by keeping catch diode and the input capacitor leads as short as possible;
- the electric radiation is minimized by reducing the area and length of all traces connected to the switch and boost pins.

Moreover, in order to reduce the noise on the feedback, that is translated in error on output voltage, the switch node and the feedback resistors are kept as far as possible from each others.

The (Fig.32) shows the *PCB* layout without ground plane (Fig.32a), then with both top and bottom ground plane (Fig.32b) which help with heat dissipation and *EMI* reduction. In (Fig.32c) and (Fig.32d) the 3-D model generated using *Ultiboard* and the final real result are compared.

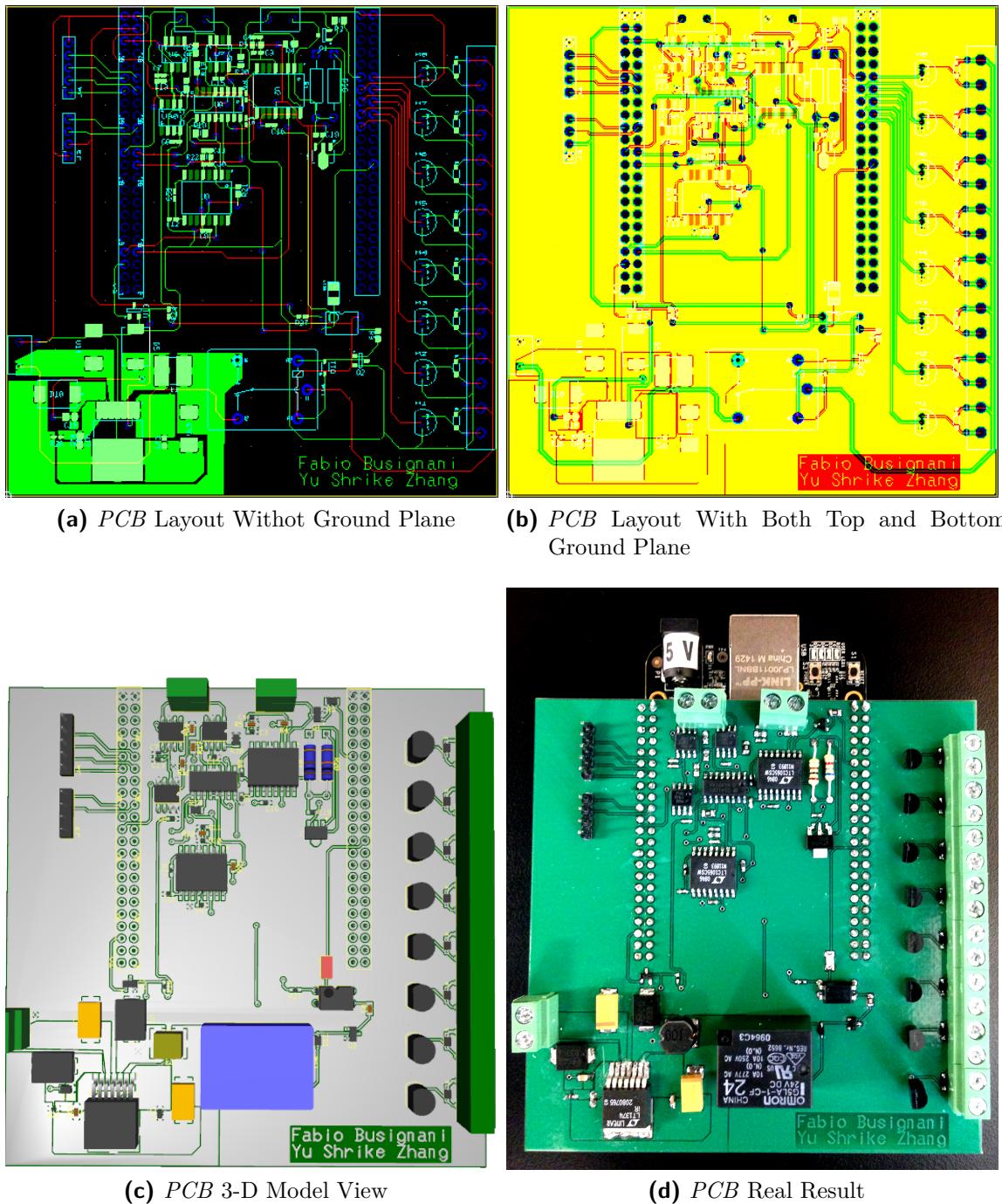


Figure 32: PCB of the System

Part II

FIRMWARE

3

INTRODUCTION

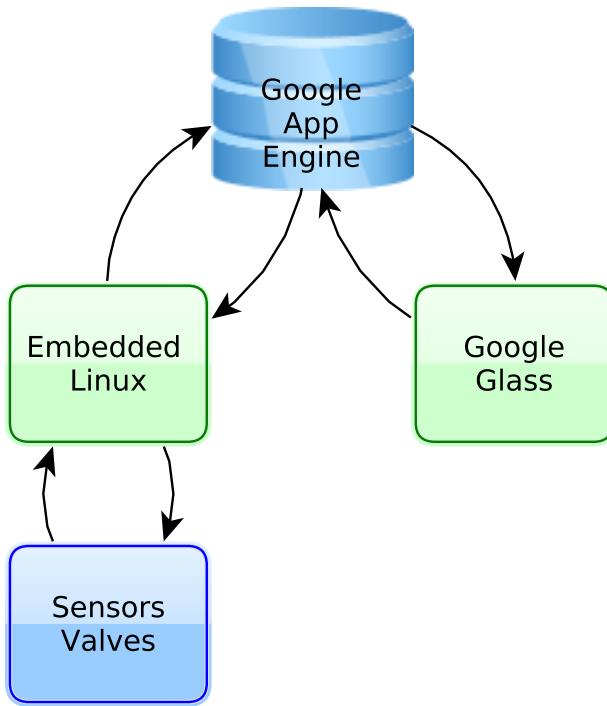


Figure 33: High Level System's Block Diagram

The term *Embedded Systems* is widely used nowadays, because its definition is very generic: any device that includes a programmable computer, but is not itself a general-purpose computer is referred as Embedded Systems.

Indeed this definition covers a huge number of systems, from a vending machine to a smartphone, form a electric toothbrush to a body computer of a car.

Peculiarity of an Embedded System is that it has hardware and software parts, takes advantage of application characteristics to optimize the design, interacts with the external environment using sensors and actuators.

Embedded Linux systems are a subset of embedded systems that are enhanced by a Linux operating system (*OS*), here the integration of high-level Linux software and low-level electronics represents a paradigm shift in embedded systems development [3]. It allows an easy way to design systems that can meet future challenges in smart buildings, the Internet of Things (*IoT*), human-computer interaction(*HCI*), robotics, and many other applications.

In this thesis project, as can be seen from (Fig.33), the system may be intended as an *IoT* one. Indeed, in order to allow the interfacing between sensors and electrovalves with the Google Glass, I had to use an Internet connection, represented by the Google App

Engine. And, while the Google Glass can interact directly with this server, the hardware described in (Part.i) needs a micro computer to be connected with that server.

3.1 A BRIEF INTRODUCTION TO IOT

The term Internet of Things is broadly used to describe the extension of the web and the Internet for the physical objects or "*things*". It is important to realize that the physical connection of the human Internet and the ones of the *IoT* are the same. And even if the *IoT* term has been coined recently, the majority of web traffic nowadays is non-human [4].

Indeed, the massive usage of online services, in part due to the smartphone revolution, has increased the interactions between servers, machine-to-machine (*M2M*) communications. This trend can only increase since it is expected an explosion of wearable systems, such as smartwatch, smartglass and so on..

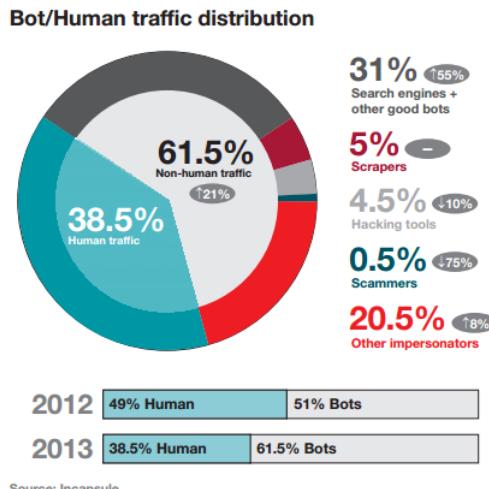


Figure 34: Distribution of Internet of Things

In the near future, *Cisco IBSG* predicts there will be 25 billion devices connected to the Internet by 2015 and this number is destined to reach 50 billion by 2020, see (Fig.35)[5]. It is also important to see that these numbers are based on the growth of 2011 and do not consider the rapid advances in device technology. So the numbers shown in (Fig.35) have to be considered as minimum.

The IoT concept, that has been exploited even in this thesis, is that if physical sensors and actuators can be linked to the Internet, then a lot of new applications and services are possible [3].

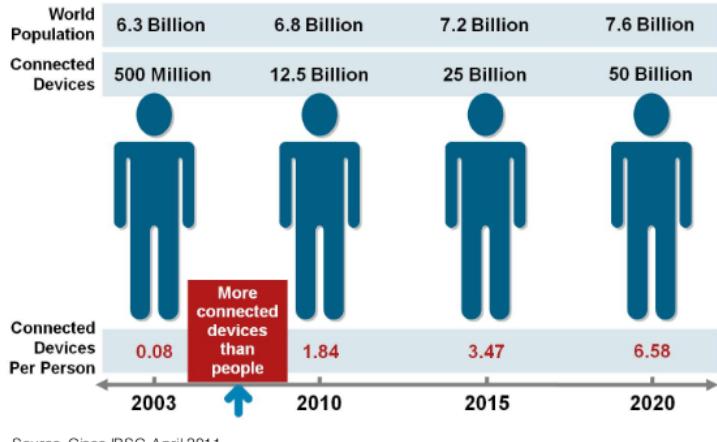


Figure 35: The *IoT* Evolution Prediction

3.2 THE BEAGLEBONE BLACK

For this thesis, the Embedded Linux system has been developed using the *Beaglebone Black*.



Figure 36: The *Beaglebone Black*

The *Beaglebone Black* is a low-cost, fully open-source (that means everyone can download and use the hardware schematics and layout in order to build up a complete product design), single board micro computer (*SBC*) powered by Linux.

There are a lot of *SBC* on the market, I chose the Beaglebone Black for the following reasons:

- low price, it costs \$50;

- powerful, the *Sitara™AM335x ARM®Cortex™-A8* processor from Texas Instruments (*TI*) can run up to 1 GHz , allowing up to 2 billion instructions per second;
- 4 GB on-board embedded multi-media card (*eMMC*), this means the Beaglebone can boot without a *SD* card, unlike other boards like Raspberry PI. It allows faster boot, approximately 10 seconds;
- 512 MB DDR3 of system memory;
- Ethernet processor, which supports *DHCP*, so it can be directly connected to a network
- **expansion headers**, 92 pins with 65 *GPIOs*, 8 analog output, 7 analog input, 3 different voltage supplies (5 V , 3.3 V , and 1.8 V), and the whole most used digital interface. All these make the Beaglebone Black built to be interfaced to.

Over these, the board has other important feature that are not used in this thesis, but may be used in the future to enhance it. An example is given by the 2 Programmable Real-time Units (*PRU*) that, unlike the definitions of Embedded Linux, make this board usable for real-time applications.

As already said in (Chap.1), this board has been connected to a custom *capes*, a daughterboard that can be attached to the two headers on the Beaglebone itself.

4

EMBEDDED LINUX INSIDE THE PROJECT

In this section the different tasks that Beaglebone has to perform are described in detail. In this project the *Linux distro* used is *Debian*. It has been chosen among the wide possibilities because ensure a good level of stability, and moreover it has a very good repository.

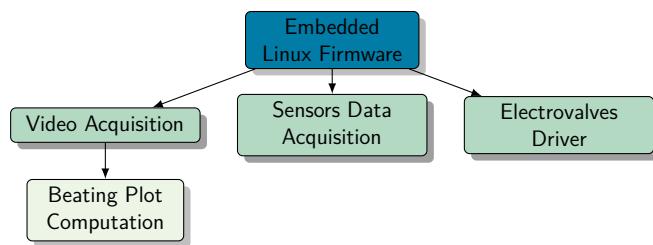


Figure 37: Embedded Linux Firmware - Blocks Subdivision

In (Fig.37) are shown the three principle subsection of the firmware. For each of them a *C++* program has been written, and their executable files are going to be run through a *bash* script, which is supposed to be launched at the system boot. The (List.A.1) shown this bash script that behaves as shown in (Fig.38).

First of all the *bash* script loads the *Device Tree Overlay*. The pins of *Sitara™* micro-processor are *multiplexed*, it means that each pin of the two header may assume different behavior (*GPIO*, *SPI*, *I2C*, and so on..). During the booting a default value for this mux is loaded, and for certain pins, this default value differs from the actual value that the system needs. In order to change those values I wrote a *Device Tree Overlays (DTO)* which has to run at the beginning, immediately after the booting, in order to explain how it work the *Flattened Device Tree (FDT)* has to be introduced.

The last releases of Linux kernels for ARM boards use *FDT*, it is a data structure that describes the hardware on board. Thus, it describes every component presents on the board, from the user *LEDs* to the *CPU*. Using this *FDT* it is possible to write a *device tree overlay* in order to change to mode of use of each pin. The (List.A.7) changes the status of 10 pins from their default value, in particular sets the pins used to drive the valves, to allow the current on resistor sensor, and to change the power supply voltage on the valves to be digital output pins, with a pulldown resistor.

To record the video from the microscope I used the program *Video for Linux v.2 (V4L2)*, before start record the *bash* script sets the video format as *MJPG* and the dimension as *320 x 240 @30 fps*. After that it runs in background the program to update the electrovalves status and to acquire the sensors value. As it is explained in (Sec.4.3), these programs run out of the infinite loop because they have an infinite loop inside.

Inside the infinite loop the *bash* script launch the recording of 10 seconds microscope video, compresses the latter and elaborate it in order to compute an image with the beating plot. Finally, it uploads (using the library *CURL*) video and image and update the flags

that in order to indicate to Google Glass App and Video Storing software that a new video has been updated.

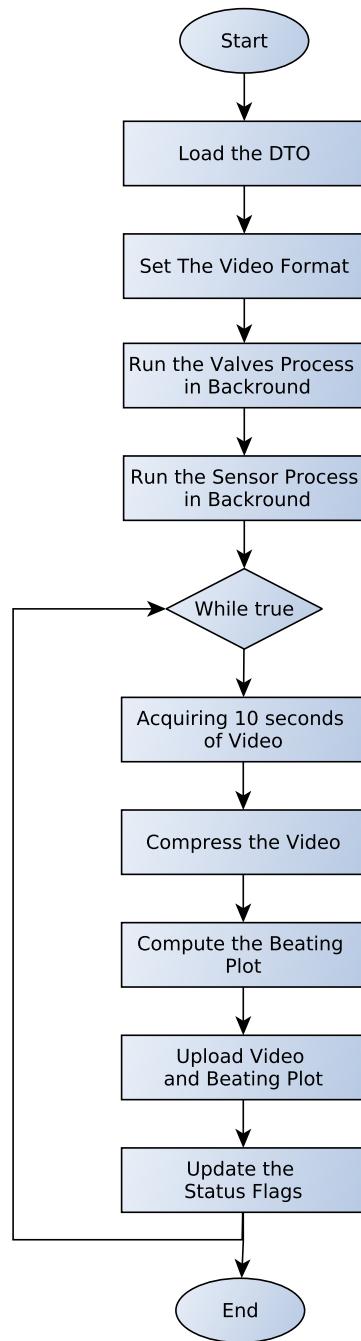


Figure 38: Bash Script Flow Chart

4.1 VIDEO PROCESSING AND BEATING PLOT

As already introduced once the microscope video has been recorded in a *MP4* format it has to be processed in order to compute the beating rate. The code described in this section is shown in (List.A.6).

To achieve this goal, the *OpenCV* library has been used. It is a specific library for computer vision.

As can be seen the program captures the video from the file and, elaborating frame by frame it computes the point to be plotted. In particular it stores the first frame of video, then each point is given by the mean different color density between the under study frame and the first one.

The results have been stored in file called *video_data.dat*. This file is going to be the input for the *gnuplot* task that is in charge to plot the beating data.

4.2 SENSOR DATA ACQUISITION

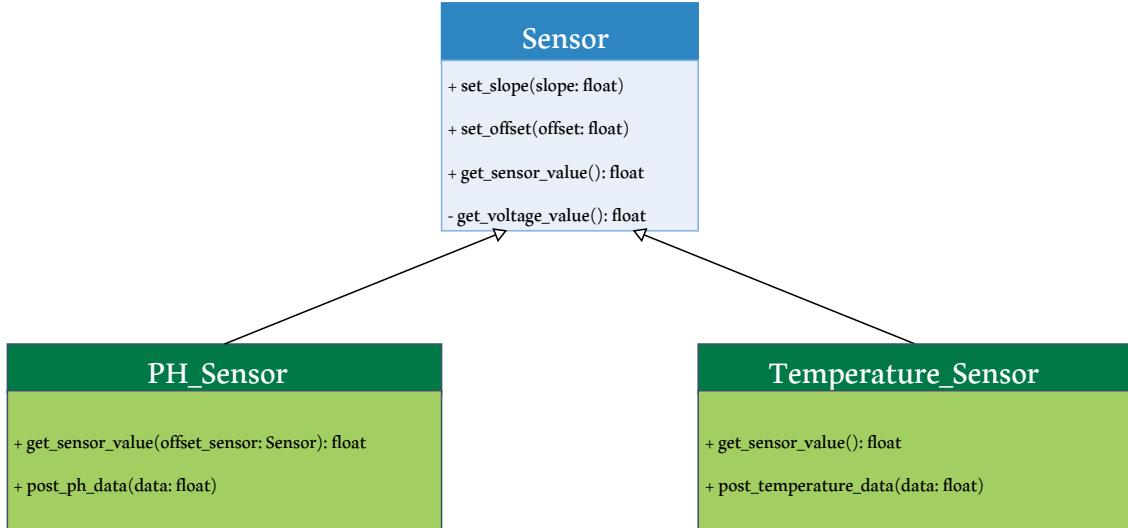
From the main *bash* script another one is launched for acquiring the sensors data. This script is shown in (list.A.2) and as can be seen it is an infinite loop where two operations are performed:

1. cleaning the previous data from the datastore of Google App Engine server, otherwise plots on the Google Glass are impossible to be viewed;
2. acquiring and uploading 100 new sensors data.

This second step is performed by another task shown in (List.A.5), in which three objects have been used (Fig.39).

How it works: using a for loop, program runs 100 times the reading of pH and temperature values, after that the reading step has been accomplished, the program is in charge to store these values on the Datastore of Google App Engine then it waits for 100 ms. Two important features have to be highlighted:

1. as already introduced in (Sec.1.3), there is a *BJT* inside the temperature conditioning circuit that acts as software-controlled switch. This is necessary to avoid the self-heating of temperature sensor and ensure a correct measure. This software-controlling may be seen from the code, indeed there is a digital output called *temperature_disable* that is normally denied, and it is asserted only when a temperature measure has to be carried out;
2. as already introduced in (Sec.1.2), the pH measuring process need to know exactly how much is the amount of added offset. Indeed, in order to make the pH sensor response unipolar, the circuit polarizes the negative pin of pH sensor at approximately 512 mV. Instead of using the theoretical value it is better to measure it just before every pH measure process. This is easily viewable from the code.

**Figure 39:** Sensor UML Description

4.3 ELECTROVALVES UPDATING TASK

The code of task which performs the driving of the valves is shown in (List.A.27). Also this program uses the *curl* library to communicate with the Google App Engine, and also this program has been built with an infinite loop.

What the task does is to download from the Google App Engine the status of each electrovalve that has been set by the user through the Google Glass. These status value are then stored onto a vector, to be written in a second step onto the output pins. As explained in (Sec.1.4) each valve is connected to a *MOS* transistor that acts as voltage-controlled switch. The voltage which drives the transistor is the digital value written onto the output pins.

The task may be called with a parameter, the supply voltage value for valves:

```
./ValvesUpdating [12 | 24]
```

If the parameter is missing, 24 V as voltage are assumed as default. This parameter is going to drive both the *DC-DC* enable and relay selector. In fact, if the voltage is the default one, the *DC-DC* does not need to work, and so, in order to reduce the power consumption, be disabled.

Part III

SOFTWARE

Rising in the logical level we find the *software*. In this part of thesis the python app which controls the server and the video storing program are going to be explained.

The first plays the important role to link the circuit, the embedded Linux system, with the Google Glass App, whenever the user is. While the second is in charge to store every new video that has been recorded, inside an hard drive of a generic computer, in order to be watched in a second time.

More details of each of two are shown in the following sections.

5 | GOOGLE APP ENGINE



Figure 40: Google App Engine Logo

Google App Engine is a Platform as a Service (*PaaS*) which allows users to build and run web applications on Google infrastructure.

Thus, when a programmer writes a web application that runs on App Engine, the software is going to run on the Google servers, somewhere in the Google cloud [6]. This solution is very useful because it allows everyone to write their own web program without purchasing and maintaining the needed hardware and network.

For the time being Google App Engine supports web applications written in a variety of the main programming languages for the web: *Java*, *Python*, *PHP*, and *GO*. Since the App Engine was first built in 2008 for Python, this is the language used to carry out the web application for this thesis.

Over the possibility to create our own web application without dealing with hardware, the Google App Engine has been chosen because of its datastore and memcache, which allow a simple memorization of data, automatic scaling and load balancing, and more others important features that are going to be shown below.

To create a more scalable and hierarchical web application the *Flask framework* has been used. Flask is a lightweight web application framework written in Python. It is based on the *WSGI* toolkit and *Jinja2* template engine and it is distributed with a *BSD* license.

Flask provides an easy way to organize a web application which allows to build up complex websites easily to manage. It has no database abstraction layer, indeed for this purpose, in this application, it supports the Google App Engine.

Before explaining how the code is made, the App Engine Datastore and Memcache are going to be introduced.

Using the Google App Engine means you do not have access to traditional databases like Oracle and MySQL. In fact, App Engine uses **Google Datastore**, which is easier to use because it takes more of a hierarchical object-oriented storage approach. That approach



Figure 41: Flask Logo

allows to ensure efficient application scalability. Thus, the Datastore holds data objects known as *entities*. An entity is composed by one or more *properties*. Making a comparison with the object oriented it is possible to say that the properties are the fields of objects. So, likely the field, a property may support different data types such as integer, float, string and so on... Any entity holds its *kind* and *key*, which categorize and uniquely identify it, respectively.

The *memcache* results very useful to speed up common Datastore queries, indeed inside the Google Datastore, distributed memory cache are used for storage. For this reason memcache is used to store the status of the electrovalves, in fact in this way the delay caused by the server is highly reduced.

5.1 THE WEB APPLICATION

The main code which runs on server is shown in (List.A.16). It uses two models to describe the data supposed to be stored:

1. *Sensor*, contains the list of sensors used from the system, through the method *sensor_list* it returns that list;
2. *DataPoint*, contains all the points that have to be plotted, the field value represent the *y – axes* while time the *x – axes*. While the first field has to be passed at the moment in which the class is constructed, the *date* field is auto-filled with the current data. This class has two public methods to interact with:
 - *point_for_sensor*, that with the name of sensor as parameter returns the list of *DataPoint* associated;
 - *oldest_point*, that returns the last *DataPoint* inserted.

Below the different services offered by this web application are listed:

- *process_values*, available at the path `/sensor_values`. User can access to this service using both *HTTP GET* and *POST* methods. With the first one, it returns the *JSON (JavaScript Object Notation)* containing all the data regarding the whole sensors stored inside the Datastore, in this format: `{ value, timestamp, sensor_name }`. On the other hand, when the used method is *POST* the application takes from the POST parameters those named "*sensor*", indicating the sensor's name, and "*value*", which is the sensor's value that has to be plotted. Thus, in this last way what the

application does is to add a new data point for a sensor and store it inside the Datastore.

- *print_names*, available at the path `/sensor_names`. This function is accessible only through HTTP *GET* method and returns a *JSON* object with the content of sensor's name list.
- *graphing_data*, available at the path `/graphing_data`. To access to this function a HTTP *GET* method is required. This *GET* request must contain two parameters: "*sensor*", the sensor name, and "*first_timestamp*", which is the timestamp of the first point that has to be plotted. This last parameter is not mandatory and if it is missed it is assumed equal to 0. What this function does is to return a *JSON* object containing the whole *DataPoint* for the given sensor that have timestamp higher than "*first_timestamp*".
- *clear_data*, available at the path `/clear`. This function is in charge to deletes all the points for each sensor.
- *set_picture*, available at path `/picture/submit`. This function is accessible with both HTTP *POST* and *GET* methods, and in both cases its behavior is to load a new image, containing the beating plot, onto Datastore. In the first case the image has been passed as *POST* data, and named as "*Image.jpg*". While the second case is supposed to be a browser way to upload the image. In fact, browsing to that *URL* what the user sees is shown in (Fig.42), a view where it is possible to select the picture file and submit it.

Upload microscope file

No file chosen

Figure 42: Picture Submitting from the Browser

- *view_picture*, available at the path `/picture/view`. Using this function, with a HTTP *GET* method causes the access to the image previously stored inside the Datastore.
- *set_video*, available at the path `/video/submit`. As happened to the beating plot image, this function is accessible with both HTTP *POST* and *GET* methods. And, as before, it is used to load onto Datastore the microscope video. Using the *POST* method means storing the data passed as parameter and named "*Video.mp4*". On the other hand, in case of GET method what the user sees on the browser is shown in (Fig.43). As before, through the two buttons, user can choose the video to store and submit the request.
- *view_video*, available at the path `/video/view`. This function, accessible with HTTP *GET* method, accesses to the microscope's video on the Datastore and returns it.

Upload microscope video

No file chosen

Figure 43: Video Submitting from the Browser

- *add_electrovalve*, available at the path `/add/electrovalve`. This function is in charge to store the valve's status inside the memcache. It is accessible through HTTP *POST* method, which must have these two following parameters: "*name*", that uniquely identifies the valves (it is "EV" followed by the number of the valve, for example the first one is "EV1"), and "*status*", which indicates the actual status of the valve (it may be *on* or *off*).
- *get_electrovalve*, available at the path `/electrovalves/<name>`. It is accessible through HTTP *GET* method and returns the value of the valve identified with *<name>* field inside the *URL* (for example, if someone wants to read the status of the first valve, has to use the path `/electrovalves/EV1`).

6

MICROSCOPE VIDEO STORING

An important role of an experiment in organ-on-a-chip applications, as well as the whole biomedical field, is given by the video analysis. In fact, analyze the video just one time in real-time, during the experiment is running is never sufficient. So, what the system described in this thesis needs to be usable, is a way to memorize the video in order to be used, watched, in a second time.

To fulfill this aim I designed a console application using the *Qt* framework. The motivation that brought me to use this framework is that the same code can run in different platform. In other words, this application can run on *Linux*, *Windows*, and *MacOS* indistinctly. This is a big advantage, because in a laboratory environment there are many researchers, and it is very easy to encounter different operating systems.

In (App.A.2) the listings of this application are shown. As can be seen, the source codes are divided in such a way to ensure high hierarchical efficiency.

Indeed, the main (List.A.10) of this application just instantiates a *MyTimer* object. This *MyTimer* object (List.A.12) is in charge to generate an interrupt every 10 sec, and it starts from its creation. When this interrupt comes, the timer uses the *Downloader* object to check if a new microscope video has been uploaded, and if so, download it and reset the flag that points up the new video status. The code of *Downloader* is shown in (List.A.14). The videos are stored inside the directory *C:/Video* and their names correspond to the date and hour of download.

The (Fig.44) shows the basic step in which the microscope video is stored. Once the Beaglebone Black has recorded the microscope video, this is upload onto Google App Engine by the board itself, and then the *video flag* is asserted in order to advice the Video Storing application that a new video has been uploaded. Finally, the application downloads and stores this video inside the computer hard drive and resets the flag.

The (List.A.11) shows the taken decision to do not use the graphic user interface (*GUI*) for this application, in order to lighten the application itself, and to allow the use of network.

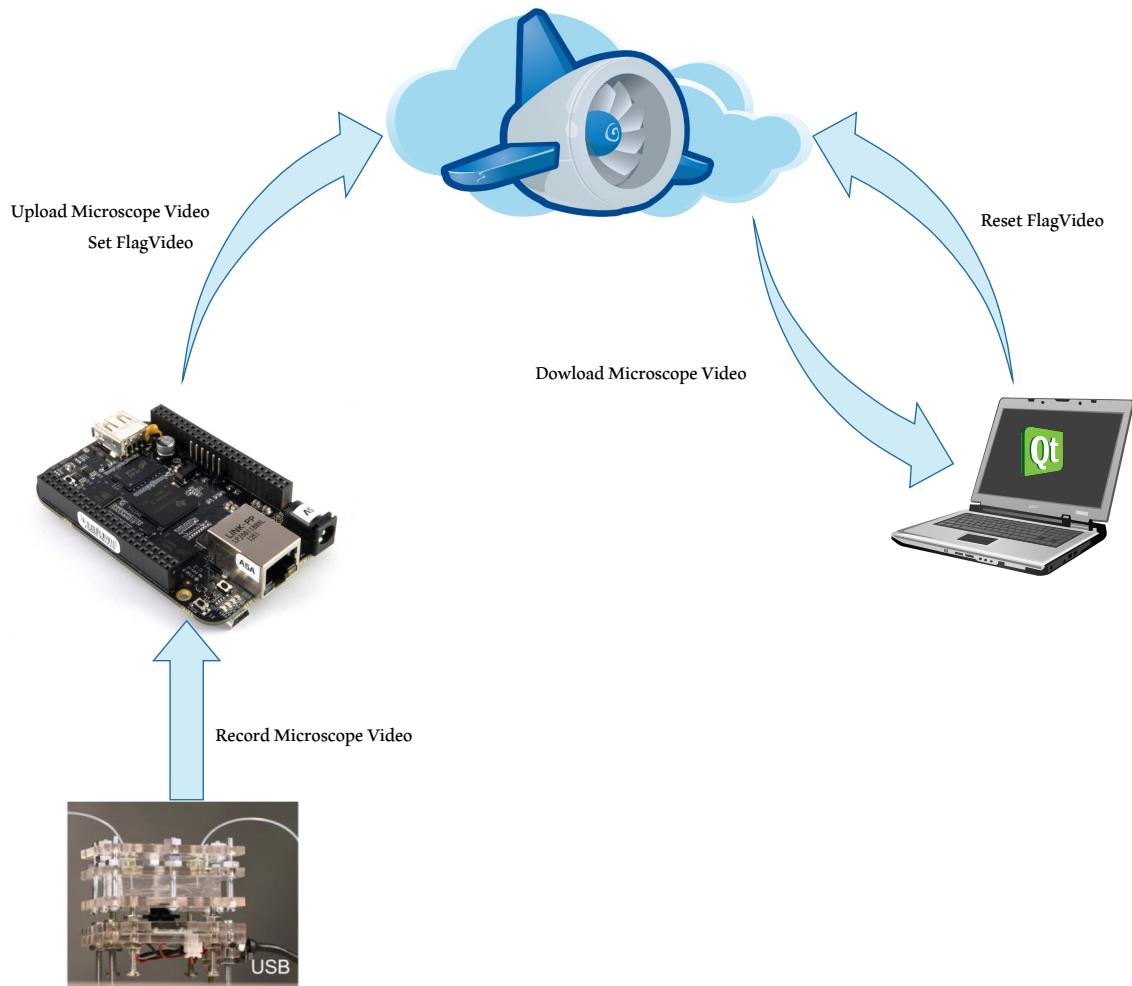


Figure 44: Video Storing: Block Diagram

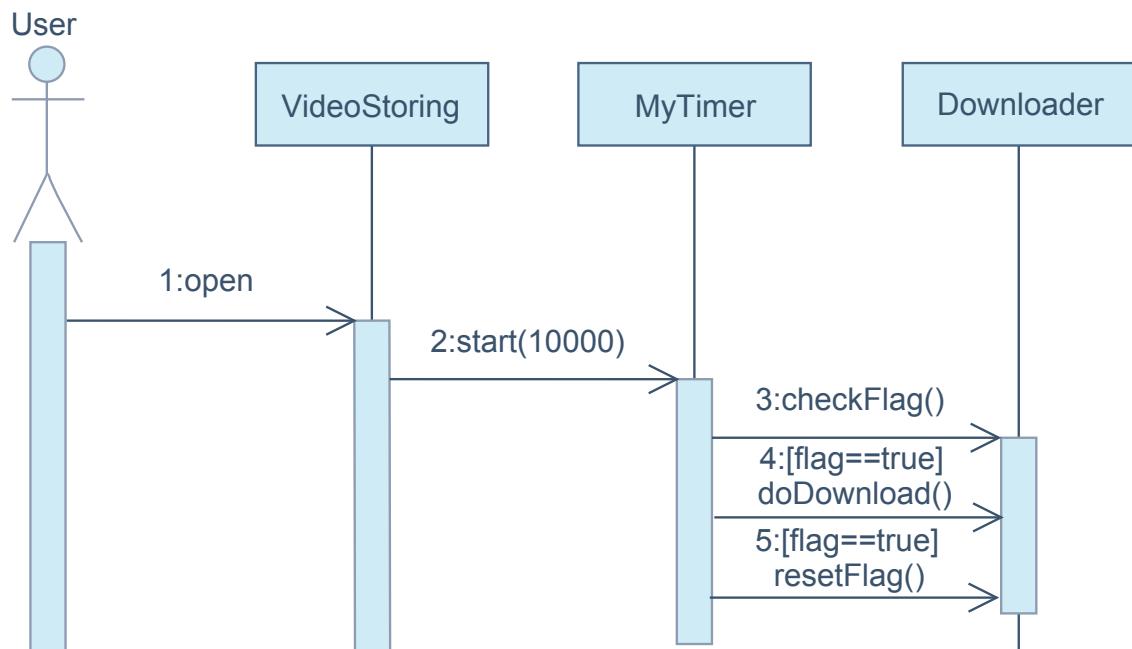


Figure 45: Video Storing: Sequence Diagram

Part IV

GOOGLE GLASS APPLICATION

7

WEARABLE COMPUTING - GOOGLE GLASS

In a world where the electronic technology is growing up faster than every other technology, and where portable devices are strongly common used by people, it is possible to be impressed by a new designed device. Looking to the science fictions, such as *Star Trek*, it was predictable that sooner or later wearable computing is going to be part of everyday life, and so it is happening. Further, this seems to be the next step of the process in how we use computers (Fig.46), going from desktop usable in a fixed location only, to portability of laptops usable and connected everywhere thank to the wireless connectivity, passing to smartphones and tablets which offer a more portability, to finally wearables computing that, for the time being, offer the highest portability [7].

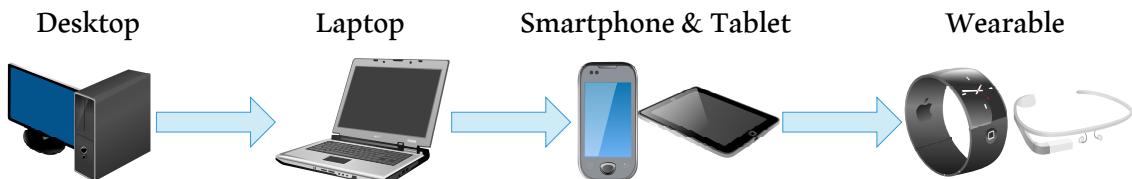


Figure 46: Computing Evolution

Thus, *smartglasses* represent an emerging space and not only Google is trying to exploit it but also other famous company like Microsoft and Sony.

7.1 GOOGLE GLASS

The Google Glass hardware is listed in (Fig.47), and we can find almost all the technology installed in a smartphone, in fact Glass is made with: a battery, a micro-USB port which has the multiple functions of power source, headphone connector, and data port, a bone conduction transducer audio (*BCT*) speaker, a touchpad, different sensors such as the accelerometer and gyroscope to detect head movements, WiFi and Bluetooth interface module, a camera, a microphone, and a prism which acts as display.

Google Glass is not an immersive experience, but it is on only when the user wants it, by watching on the top right corner, where the 640×360 display is placed, and says "Ok Google" or tapping onto touchpad.

The Google Glass applications are called **Glassware** and they are available in two different way: Google Mirror API and Glass Development Kit (*GDK*).

The *timeline* is a chronological list of cards, each card represents a Glassware, does not matter which kind of it is. In other words, timeline is a way to organize the opened application, and users scrolling through different section of the timeline is able to reveal cards in the past, present, and future.

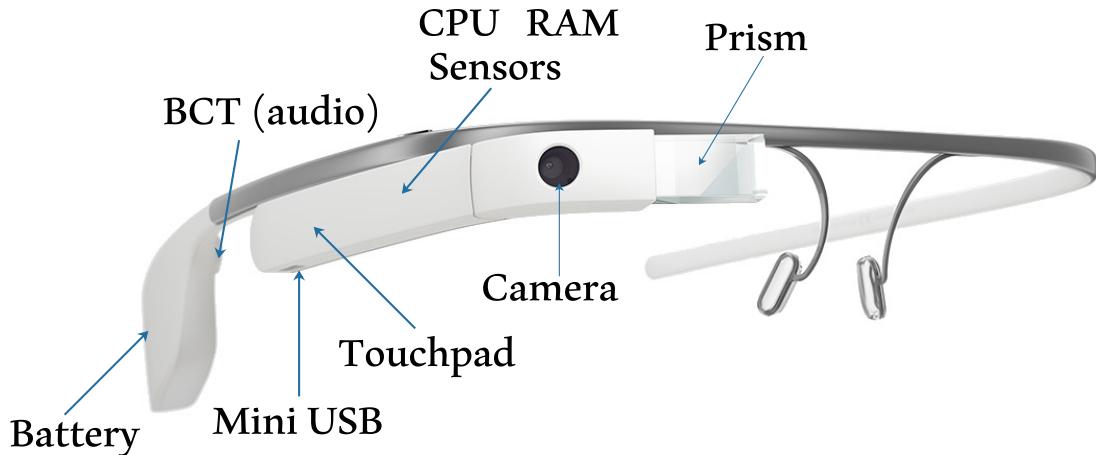


Figure 47: Google Glass

7.1.1 Mirror API

Mirror API Glassware is a web-based application. In fact, the code does not run on the Google Glass itself, but it runs purely in the cloud provided by Google. In this way many programming languages are available: Java, Python, PHP, Ruby, .NET and Go.

The main advantage of this applications is that, because the Glassware does not run on Glass, its processor is left free to work on other things and it is not in charge to make some data manipulation, calculation and so on..

This service has two big requirements: of course, being connecting and cloud-aware. Mirror API Glassware has no executable file, and has no access to the Glass sensors.

7.1.2 Glass Development Kit

Glassware built using the Glass Development Kit offers more granular control of application. The programming language available is only one: Java.

This may be defined the classic Android way, where Java code is built in an *APK* (Android Package) file and installed inside the Glass device.

GDK based Glassware has more functionality than the Mirror API one, indeed it may directly access the hardware, ensure real-time interactions, and be able offline.

The Glassware designed in this thesis is GDK based.

8 | THE GLASSWARE

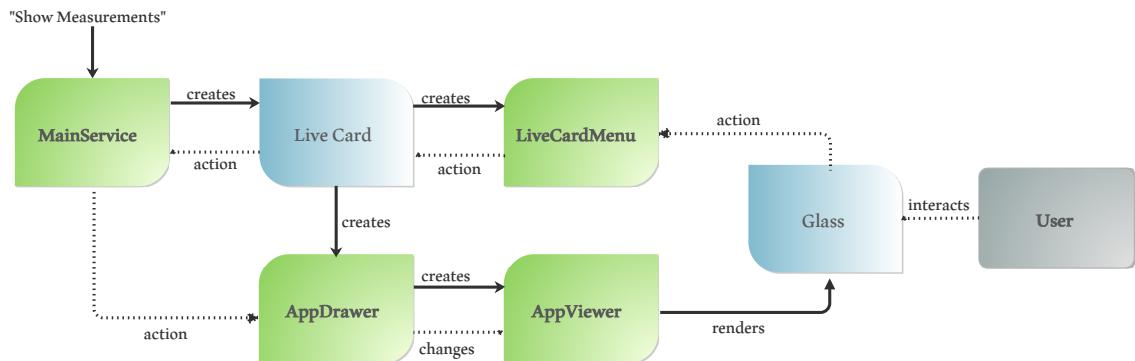


Figure 48: Interactions Between Class

In (Fig.48) how the classes and actors in the designed Glassware interact from each others is shown. This is also how a typical *GDK* based Glassware works.

It stars with the service called *MainService* (List.A.17) that runs when the Google Glass voice command is triggered or the application is tapped from the Google Glass Glassware list. When the service has been started, first it creates a live card and then it runs four runnable in background, (Fig.49):

- *Sensors Values*, a runnable that periodically downloads and updates the pH and temperature sensors values which then will be plotted.
- *Video*, a runnable that periodically checks if a new microscope video is available and, if so, downloads this video inside the Google Glass replacing the older one.
- *Beating Image*, a runnable that periodically downloads the beating plot replacing the older one.

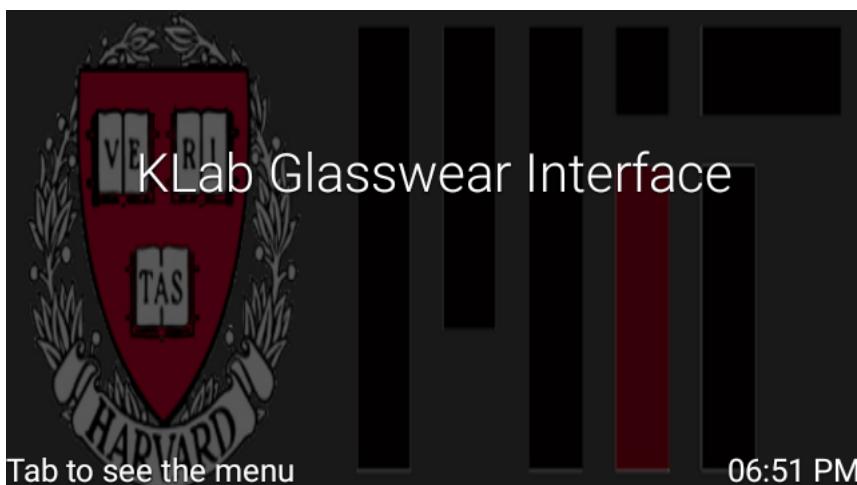


Figure 49: MainService's Runnable in Background

- *Electrovalves Status*, a runnable that periodically downloads the status of each valve.

A *live card* is one of the most important element in the *Google Glass User Interface (UI)*, it appears in the present section of the timeline and shows information that is relevant at the current time. Live cards can access low level Glass hardware, such as camera, sensors, communication modules, and so on...

A live card persists in the present section of the timeline as long as the user thinks it may be relevant. Indeed, it is not persistent in the timeline and user can explicitly remove it through a swipe down on the touchpad. Further, more than one live cards may run at the same time, in this case, a live card is still running even if not visible and the focus is on another one.

Live cards are divided in two categories:

1. *Low-Frequency Rendering*, limited to a small set of views and can only update the display once every few seconds.
2. *High-Frequency Rendering*, involve content that is update frequently, more time in a second. This is very useful for plotting data coming from sensors, exactly like the application in which this Glassware is involved, this is why this kind of live card had been used in this thesis project. To create this type of live card an inflating layout has been used.

Finally, live cards exist as long as the Android service which statically generated it is running.

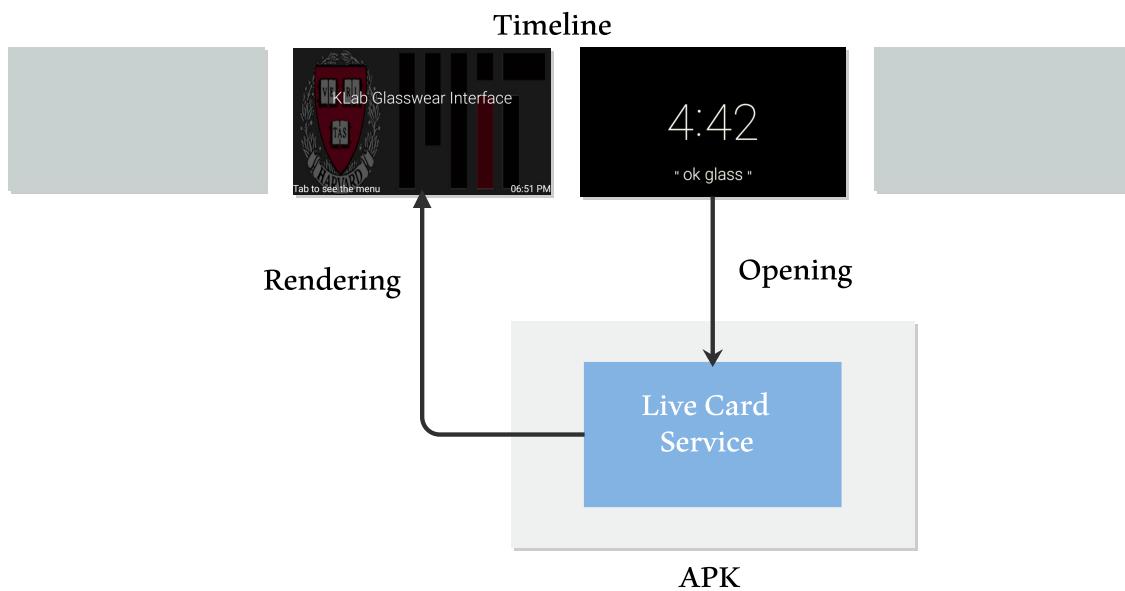


Figure 50: Live Card, Adding to the Timeline

Backing to the Glassware architecture, once the live card has been created, it constructs an instance of the class *AppDrawer*, an implementation of *Callback* for rendering and thus managing the Glassware *UI*. Live card also adds an instance of the *LiveCardMenu* class, which dynamically manages the menu of the Glassware. Indeed, the application menu changes based on the actual view displayed on the live card.

Then *AppDrawer* creates instances of View subclass, the number of them is four: pH plot, temperature plot, beating plot and electrovalves status. In the (Fig.48), these four instance are not shown for compact reasons, instead of them a generic *AppViewer* class is present.

AppViewer performs the real rendering, setting the layout and drawing the content on that.

Always in (Fig.48) with continuous lines, the steps made when the Glassware has just been opened are shown, while the dashed lines highlight the interactions, that take part when the user acts on Glass.

8.1 THE CLASSES

Below the different classes that compose the Glassware are going to be shown.

8.1.1 Main Service

The code of the main service class is displayed in (List.A.17). As already said, this class is in charge to create the live card, and to it the live time of the Glassware itself is tied. So, this happens when the service is started at the *OnStartCommand*. After that, an *AppDrawer* instance is created and set as callback in order to render the live card surface.

The *OnStartCommand* method also creates the runnable tasks which run in background to update the data information coming from the Google App Engine Datastore

Finally the main service is also in charge to handle the user requests, that are passed through the menu activity.

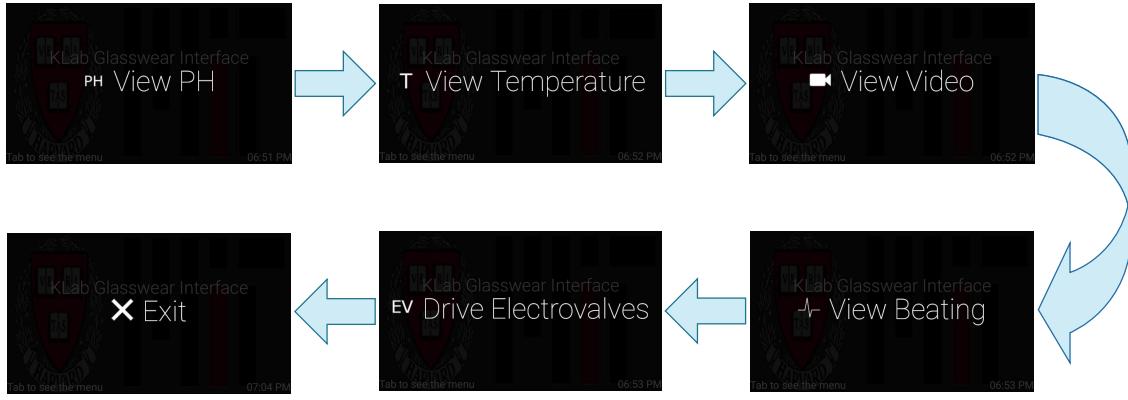
8.1.2 App Drawer

The code of the App drawer is shown in (List.A.18). This subclass allows the control of display surface and chooses which View as to be set at a given time. In fact, it first creates an instance of each view subclass, with their *listener* interface. Then implements the listener method, this is a convenient way to skip from a view to another one. Finally it runs the view draw request, locking the surface canvas.

8.1.3 Menu Activity

The code of the menu activity is shown in (List.A.24). This subclass plays a very relevant role in the Glassware functionality. Indeed, it shows the different chooses that allow the user to navigate into the application.

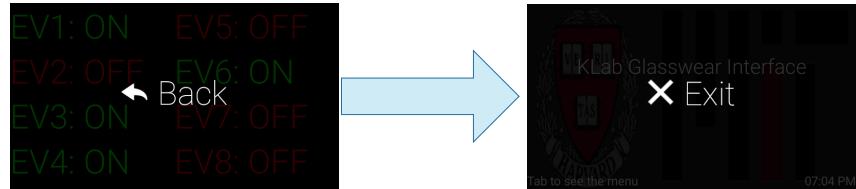
To fulfill that aim it has to change dynamically, based on what is the viewer on focus of the live card. From the main view, user can choose one of the following options: *View pH*, *View Temperature*, *View Video*, *View Beating*, *Drive Electrovalves*, and *Exit*.

**Figure 51:** Main Menu

To navigate the menu, user must scroll with a finger, swiping on the touchpad.

The image in (Fig.51) shows the menu options from the Glassware main view. The (Fig.52) shows the menu options when the graphs (pH, temperature, and beating) are displayed.

Finally, (Fig.53) shows the menu options when the *Drive Electrovalves* is displayed. In that figure only four choices of eight are shown, for compact reasons only. To toggle the valves the Glassware uses two classes: Electrovalves, static class where the status of the valves is stored, and HTTP subclass of Android service, used to send HTTP *GET* and *POST*. The parameter of the HTTP request are attached to the intent that creates the service.

**Figure 52:** Graph Menu**Figure 53:** Menu Electrovalves

Thus, the menu activity is in charge to show dynamically the possible options as well as handle the requests using the Android *intents* to inform the other subclasses what they are supposed to do.

8.1.4 App Manager

The code of this class is shown in (List.A.25). The *App Manager* is a static class that is in charge to memorize the state, what is the view that has to be displayed.

8.1.5 Main View

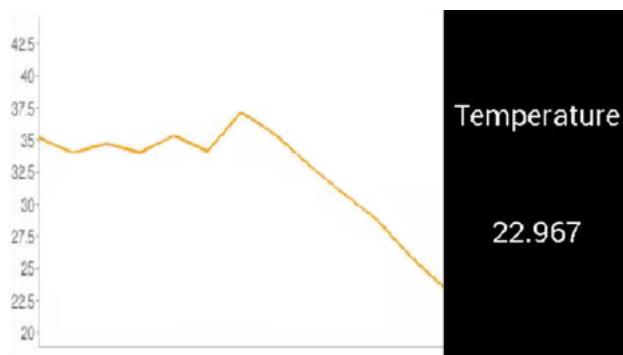
The code of this class is shown in (List.A.19). The *Main View* instantiates the layout *XML* file and sets the Harvard-MIT logo as background image, "KLab Glassware Interface" as main written, "Tab to see the menu" as left footer, and the current hour as right footer.



Figure 54: Main View

8.1.6 PH and Temperature Views

Their codes are shown in (List.A.21) and (List.A.22). As for the main view, also this subclass instantiates a layout *XML* file and plot the data, write the kind of sensor is it ("Temperature" or "pH") on the top, and the value of the last data point on the bottom.



Figure

8.1.7 Beating View

The last plot, the beating one, is shown in (Fig.56). The code of this subclass is listed in (List.A.20). In this case the layout is made by an *ImageView* only, and the subclass just set the stored beating plot image.

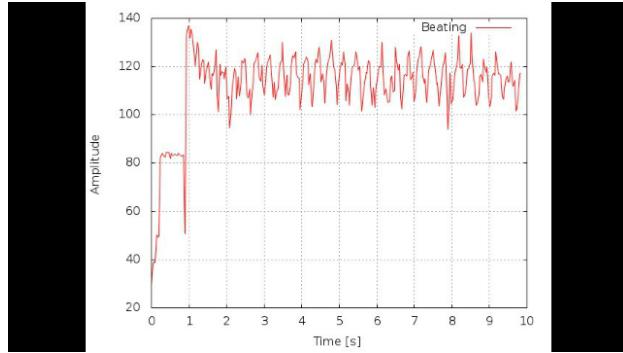


Figure 56: Beating View

8.1.8 Electrovalves View

EV1	EV5	EV1: OFF	EV5: OFF
EV2	EV6	EV2: OFF	EV6: OFF
EV3	EV7	EV3: OFF	EV7: OFF
EV4	EV8	EV4: OFF	EV8: OFF

(a) When Status Has Not Dowloaded Yet

(b) When They Are All Off

EV1: ON	EV5: OFF
EV2: OFF	EV6: ON
EV3: ON	EV7: OFF
EV4: ON	EV8: OFF

(c) When Some of Them is On

Figure 57: Electrovalves View

The (Fig.57) shows the result of the electrovalves view. It lists all the eight valves showing the corresponding status. When the user opens the *Drive Electrovalves* tab and the background tasks has not downloaded the valves status yet, the whole electrovalves

are displayed without their status, and white written, (Fig.57a). When the valves status has been downloaded, the name and the status appear in green color if the valve is on, or in red if it is off, (Fig.57b) and (Fig.57c).

8.1.9 Video Activity

To play the stored video on Google Glass there are two way, the first one uses the `VideoView` object, and the second one uses an intent belonging to Google Glass API: `com.google.glass.action.VIDEOPLAYER`. The Glassware described so far exploits this second way, because so the video player is optimized for the Google Glass. Indeed, user can navigate into the video with simple horizontal swipes on the touchpad (Fig.58a), can stop the video with a simple tap (Fig.58b), and to close the video uses a vertical swipe.

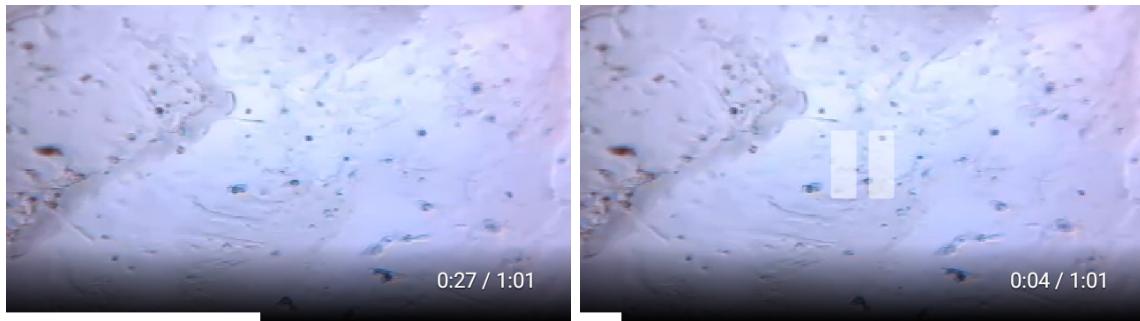


Figure 58: Microscope Video

Part V

CONCLUSION AND APPENDIX

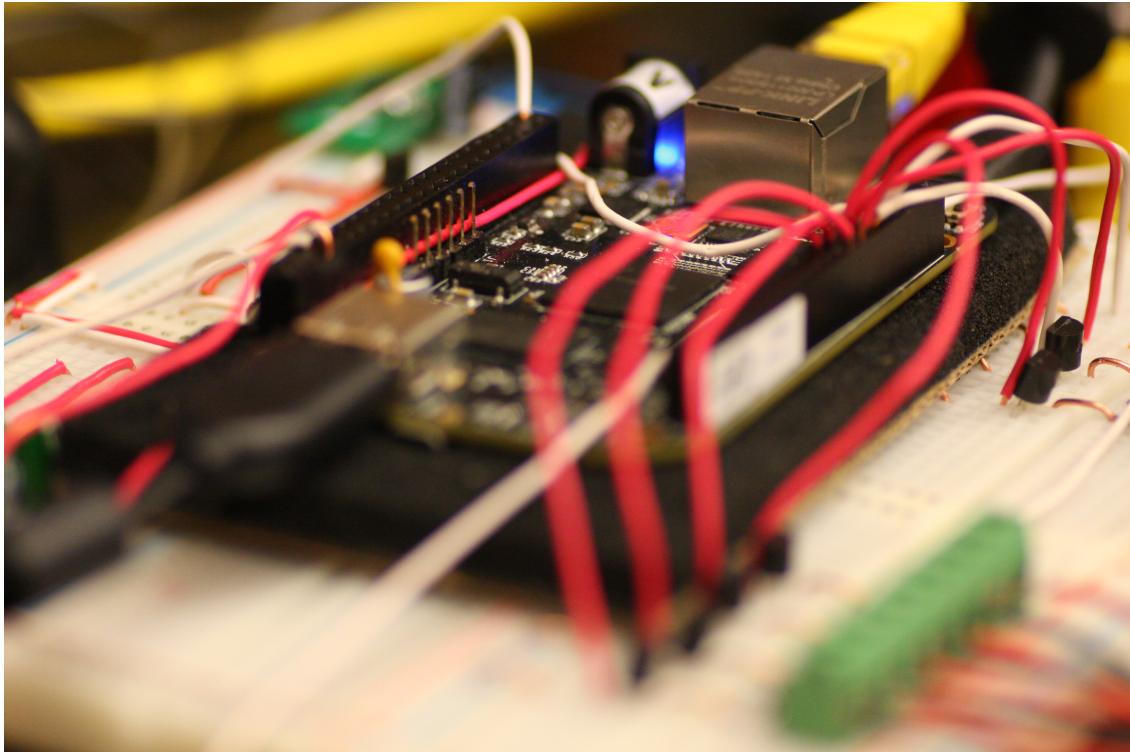


Figure 59: Circuit mounted on breadboard side view

For what concern the sensors acquiring paths, unfortunately, rigorous experiments have not been done. This is due to the end of my six months period. What I can say for those parts is that from the experiment that I made (not in a microfluidic or biomedical context) the results appear acceptable.

While, in order to try the reverse control from the Google Glass to the electrovalves we made different kind of experiments.

9.1 LED EXPERIMENTS

First of all we tried the circuit on a breadboard using LEDs instead of electrovalves. The aim of this step is to demonstrate that the firmware running on the Beaglebone Black, the Java code running on the Google Glass and the Python code running on the Google App Engine (used to store the information about the electrovalves status) work well. Moreover the LED and the electrovalve have basically the same behavior so, if everything works well with the LEDs, there are all the reasons to believe that everything is

going to work well with the electrovalves, too.

The circuit that actually drives the LEDs is very simple, and it is based on a MOS transistor (*BS170*) used as a switch voltage-controlled, as shown in (Fig.60).

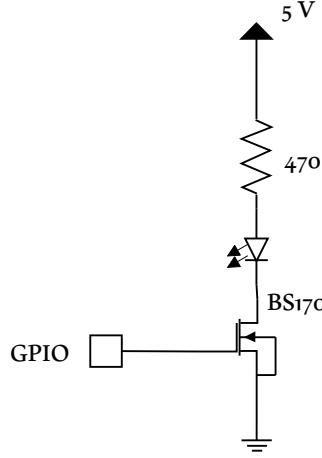


Figure 60: Driver for LED

The (Fig.61) shows the circuit used for this step of testing. As can be seen the number of LEDs used is eight, the same number of electrovalves that can be driven from this system.

In order to test all of them we made 2 different kind of trials:

1. *In order turning on&off*, first all the LEDs are turned on starting from the first one (on the top right corner) to the last one (on the bottom left corner). Then the LEDs are turned off following the same order.

The result of this can bee watched in [this](#) video.

2. *Out of order turning on&off*, in this trial, like before, all the LEDs start from a condition where all of them are off and then we turned on and off all the LEDs, but in this case following a random order.

The result of this can bee watched in [this](#) video.

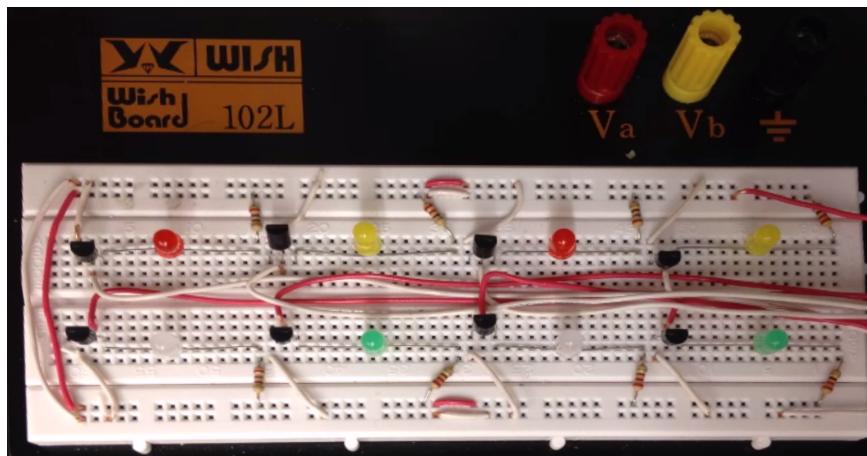


Figure 61: LEDs experiments board

9.2 ELECTROVALVES EXPERIMENTS

9.2.1 Breadboard Phase

The (Fig.62) shows from the top view the circuit used during the second phase of experiment, the one where we started using electrovalves in a real microfluidic application. On the left side of the figure we can see the conditioning circuits for the temperature sensor (on the top) and pH sensor (on the bottom). While, on the other side, we can see the part of circuit in charge to drive the electrovalves.

In this last one we are going to focus for now. Each electrovalve is driven by the circuit shown in (Fig.23).

As you can see, this circuit is pretty close to the one of (Fig.60), indeed the only difference is given by freewheeling diode, mandatory because of inductive behavior of electrovalve's solenoid.

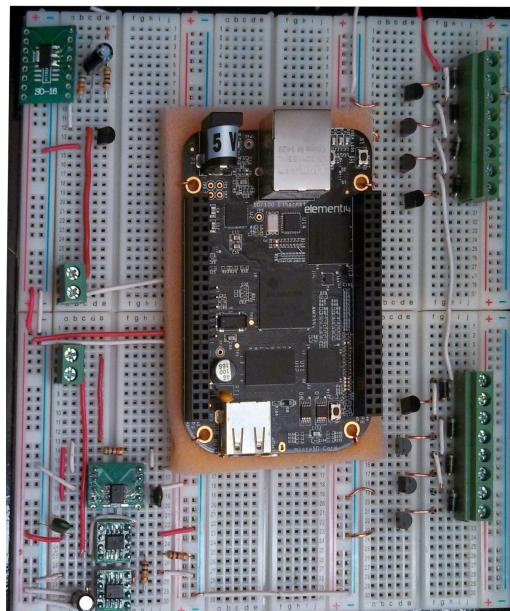


Figure 62: Circuit mounted on breadboard top view

The result of the experiments with electrovalves in a real microfluidic case can bee watched in [this video](#).

9.2.2 PCB Phase

Finally we replied the last experiment using a PCB (Fig.32), designed for this system. As expected the result of this experiment is the same of the previous step.

10 | LIMITATIONS AND IMPROVEMENTS

As already said, unfortunately, the experiments made to test the sensors acquiring have not be made in a biomedical context.

What I did was, in a first time, to test whether the Glassware is able to plot dummy data stored inside the Google App Engine. And in a second time, I tried to plot data from pH and temperature sensors, but not in an organ-on-a-chip application. In both cases the results of experiments were positive, in any case I strongly suggest experiments in a biomedical context.

The embedded system that has been developed involves Internet connection and a server. every time that we are talking about Internet of Think, it is important to deal with Internet security. Thus, the next step of this project regards how to keep the data safe, and how to ensure the privacy, in such a way that no one which is not allowed to see data can have access to them.

A first, and fast way to fulfill this aim is using accounts. In fact, *Flask* framework gives the possibility to hide some link if the user is not logged in the web application. In this way, the *Qt* application, the Glassware, and the Embedded Linux firmware have to log-in as the first step.

As is shown in (Fig.7), *PCB* presents two connectors that are not used yet: *Buttons* and *Display* connectors. As can be seen fro the circuit schematic, (Fig.29), the first connector has its pins tied to *GPIO* pins of the microcontroller and it is supposed to be used in order to add 4 additional buttons. While the second one is tied to *I2C0* interface of the microcontroller and is though to connect a *I2C display*. In this way, the circuit allows some additional applications that do not require the Google Glass and PC interactions. For example a possible, and really useful, application is to include the sensors calibration. Indeed, for the time being, user has to make the calibration and then modify the bash script which runs the sensor acquiring, (List.A.2):

```
./sensor_acquiring [slope_temperature slope_ph  
offset_temperature offset_ph]
```

While embedding the sensors calibration, user must perform the same calibration steps as before, but the linear regression and transfer function modification are automatically performed by the system.

A | CODE

A.1 FIRMWARE

Listing A.1: KlabFirware (bash)

```
#!/bin/bash
2 echo "Load the DTO"
echo GlassProject-GPIO > $SLOTS
4
echo "Setting the video format as MJPG"
6 v4l2-ctl --set-fmt-video=width=320,height=240,pixelformat=1 -d 0
v4l2-ctl --set-parm=30
8
echo "Running the Electrovalve status updating task"
10 ./Electrovalves/ValvesUpdating </dev/null &>/dev/null &
./Sensors/sensorAcquiring </dev/null &>/dev/null &
12
while true; do
14   echo "Recording ten second of Video"
   ./Video/capture -d /dev/video0 -F -o -c 300 > output.raw -y
16   avconv -f mjpeg -i output.raw -vcodec copy output.mp4 -y
   echo "Compressing Video"
18   ffmpeg -i output.mp4 -acodec mp2 Video.mp4
   echo "Uploading the Video to the server"
20   curl -include --form Video.mp4=@Video.mp4 -A "National Instruments LabVIEW"
      http://CENSURED/video/submit -0
22   echo "Updating the flag for video"
   curl --data "name=video&status=on"
24     http://CENSURED/add/electrovalve
   curl --data "name=glass&status=on"
26     http://CENSURED/add/electrovalve
   echo "Ploting the beating rate"
28   ./Video/compute_beating
   echo "Uploading the beating plot"
30   curl -include --form Image.jpg=@Image.jpg -A "National Instruments LabVIEW"
      http://CENSURED/picture/submit -0
32   echo "finish"
done
```

Listing A.2: SensorAcquiring (bash)

```

1 #!/bin/bash

3 while true; do
4     echo "Clearing previous data on server"
5     curl http://CENSURED/clear
6     echo "Acquiring and uploading new sensors data"
7     ./sensor_acquiring

9 done

```

Listing A.3: Pins Setting

```

1 /*
2 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
3 *
4 * This program is free software; you can redistribute it and/or modify
5 * it under the terms of the GNU General Public License Version 2 as
6 * published by the Free Software Foundation
7 *
8 * Original from: github.com/jadonk/validation-scripts/blob/master/test-capemgr/
9 *
10 * Modified by Fabio Busignani fbusigna@mit.edu
11 *
12 */
13
14 /dts-v1/;
15 /plugin/;

17 /{
18     compatible = "ti,beaglebone", "ti,beaglebone-black";
19     part-number = "KLab";
20     version = "00AO";

22
23     fragment@0 {
24         target = <&am33xx_pinmux>;
25
26         __overlay__ {
27             ebb_example: KLab {
28                 pinctrl-single,pins = <
29                     0x04c 0x07 // P9_16 - T - Output Mode7 pulldown
30                     0x024 0x07 // P8_13 - EV1 - Output Mode7 pulldown
31                     0x028 0x07 // P8_14 - EV4 - Output Mode7 pulldown
32                     0x03c 0x07 // P8_15 - EV3 - Output Mode7 pulldown
33                     0x038 0x07 // P8_16 - EV6 - Output Mode7 pulldown
34                     0x02c 0x07 // P8_17 - EV5 - Output Mode7 pulldown
35                     0x08c 0x07 // P8_18 - EV8 - Output Mode7 pulldown
36                     0x020 0x07 // P8_19 - EV7 - Output Mode7 pulldown
37                     0x030 0x07 // P8_44 - EV2 - Output Mode7 pulldown
38             >;
39         };
40     };
41 }

```

```

37          0x0a0 0x07 // P8_46 - PowerSupply

39      >;
40  };
41  };
42
43  fragment@1 {
44  target = <&ocp>;
45  __overlay__ {
46  gpio_helper {
47    compatible = "gpio-of-helper";
48    status = "okay";
49    pinctrl-names = "default";
50    pinctrl-0 = <&eklab>;
51  };
52  };
53  };
54};

55}

```

Listing A.4: ElectrovalvesDriver.cpp

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <iostream>
4 #include <sstream>
5 #include <curl/curl.h>
6 #include <GPIO/GPIO.h>

7
8 #define NUMBER_ELECTROVALVES 8

9
10 using namespace std;

11
12 std::string const ELECTROVALVES_LINK =
13     "http://CENSURED/show/";
14 std::string const TRUE_VALUE = " True ";
15 std::string const FALSE_VALUE = " False ";

16 static size_t WriteCallback(void *contents, size_t size, size_t nmemb, void *userp)
17 {
18     ((std::string*)userp)->append((char*)contents, size * nmemb);
19     return size * nmemb;
20 }

21
22 int main(void)
23 {
24     // constructor of pins
25     GPIO EV1(44), EV2(23), EV3(26), EV4(47), EV5(46), EV6(27), EV7(65), EV8(22);
26     GPIO RELAY(61);
27 }

```

```
29     GPIO_DCDCENABLE(20);  
30  
31     CURL *curl;  
32     CURLcode res;  
33  
34     GPIO_VALUE output_vector[NUMBER_ELECTROVALVES];  
35  
36     //Set output pins  
37     EV1.setDirection(OUTPUT);  
38     EV2.setDirection(OUTPUT);  
39     EV3.setDirection(OUTPUT);  
40     EV4.setDirection(OUTPUT);  
41     EV5.setDirection(OUTPUT);  
42     EV6.setDirection(OUTPUT);  
43     EV7.setDirection(OUTPUT);  
44     EV8.setDirection(OUTPUT);  
45  
46     RELAY.setDirection(OUTPUT);  
47  
48     if (argc == 5)  
49     {  
50         if(atoi(argv[1])==24)  
51         {  
52             RELAY.setValue(HIGH);  
53             DCDCENABLE.setValue(HIGH); //disable the DCDC  
54         }  
55         else  
56         {  
57             RELAY.setValue(LOW);  
58             DCDCENABLE.setValue(LOW); //enable the DCDC  
59         }  
60     }  
61     else  
62     {  
63         RELAY.setValue(HIGH);  
64         DCDCENABLE.setValue(HIGH); //disable the DCDC  
65     }  
66  
67     while(1)  
68     {  
69         //acquire the values of electrovalves status and store them inside  
70         //output_vector  
71         int i;  
72         for(i=0;i<NUMBER_ELECTROVALVES; i++)  
73         {  
74             curl = curl_easy_init();  
75             if(curl)  
76             {  
77                 //Perform the curl operation  
78             }  
79         }  
80     }  
81 }
```

```

77     std::string readBuffer;
78     std::ostringstream ss;
79     ss << i+1;
80     std::string url = ELECTROVALVES_LINK + "EV" + ss.str();
81     curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
82     /* example.com is redirected, so we tell libcurl to follow redirection */
83     curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
84     curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
85     /* Perform the request, res will get the return code */
86     res = curl_easy_perform(curl);
87     /* always cleanup */
88     curl_easy_cleanup(curl);
89
90
91     if(readBuffer.compare(TRUE_VALUE) == 0)
92     {
93         std::cout << "EV" + ss.str() + ": ON" << std::endl;
94         output_vector[i] = HIGH;
95     }
96     else if(readBuffer.compare(FALSE_VALUE) == 0)
97     {
98         std::cout << "EV" + ss.str() + ": OFF" << std::endl;
99         output_vector[i] = LOW;
100    }
101}
102
103 //now I'm ready to write the output pins:
104 EV1.setValue(output_vector[0]);
105 EV2.setValue(output_vector[1]);
106 EV3.setValue(output_vector[2]);
107 EV4.setValue(output_vector[3]);
108 EV5.setValue(output_vector[4]);
109 EV6.setValue(output_vector[5]);
110 EV7.setValue(output_vector[6]);
111 EV8.setValue(output_vector[7]);
112 }
113 }
```

Listing A.5: sensor_aquiring.cpp

```

1 #include<iostream>
2 #include<fstream>
3 #include<unistd.h>
4 #include<string>
5 #include<sstream>
6 #include<stdlib.h>
7 #include<curl/curl.h>
8 #include"../GPIO.h"
9 #include"../http.h"
```

```
using namespace std;
11 #define LDR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"
12 #define GPIO_PATH "/sys/class/gpio/"
13 #define OUTPUT false
14 #define INPUT true
15 #define HIGH true
16 #define LOW false
17
18 float const CURRENT = 0.00068;
19 float const SLOPE_TEMPERATURE = 2;
20 float const OFFSET_TEMPERATURE = 1870;
21 float const SLOPE_PH = -0.070;
22 float const OFFSET_PH = 0.6;
23
24 class Sensor
25 {
26     int number;
27
28 public:
29     float m;
30     float q;
31     Sensor(int x): number(x){}
32     void set_slope(float slope){
33         m = slope;
34     }
35     void set_offset(float offset){
36         q = offset;
37     }
38     float get_sensor_value(){
39         float voltage_value = this->get_voltage_value();
40         float sensor_value = (voltage_value - q)/m;
41         return sensor_value;
42     }
43
44 private:
45     float get_voltage_value(){
46         stringstream ss;
47         ss << LDR_PATH << number << "_raw";
48         fstream fs;
49         int adc_value;
50         fs.open(ss.str().c_str(), fstream::in);
51         fs >> adc_value;
52         fs.close();
53         float cur_voltage = adc_value * (1.80f/4096.0f);
54         float actual_value;
55         return cur_voltage;
56     }
57 };
};
```

```

59   class Temperature_Sensor : public Sensor{
61     private:
63     public:
64       Temperature_Sensor(int x) : Sensor(x) {}
65       float get_sensor_value(){
66         float voltage_value = this->get_voltage_value();
67         float resistor_value = voltage_value / CURRENT;
68         float sensor_value = (resistor_value - q)/m;
69         return sensor_value;
70     }
71     void post_temperature_data(float data){
72       post_data_sensor(0, data);
73     }
74   };
75
76   class PH_Sensor : public Sensor{
77     public:
78       PH_Sensor(int x) : Sensor(x) {}
79       float get_sensor_value(Sensor offset_sensor){
80         //Sensor offset_sensor(2);
81         float voltage_value = this->get_voltage_value();
82         q += offset_sensor.get_voltage_value();
83         float sensor_value = (voltage_value - q)/m;
84         return sensor_value;
85       }
86
87       void post_ph_data(float data){
88         post_data_sensor(1, data);
89       }
90     };
91
92
93   int main(int argc, char* argv[])
94   {
95     GPIO temperature_disable(45);
96     float slope_temperature;
97     float slope_ph;
98     float offset_temperature;
99     float offset_ph;
100
101    if (argc == 5)
102    {
103      slope_temperature = atoi(argv[1]);
104      offset_temperature = atoi(argv[2]);
105      slope_ph = atoi(argv[3]);
106      offset_ph = atoi(argv[4]);
107    }

```

```

109     else
110     {
111         slope_temperature = SLOPE_TEMPERATURE;
112         offset_temperature = OFFSET_TEMPERATURE;
113         slope_ph = SLOPE_PH;
114         offset_ph = OFFSET_PH;
115     }
116
117     //set the pin GPIO_45 as output
118     temperature_disable.setDirection(OUTPUT);
119     //Temporary disable the temperature
120     temperature_disable.setValue(HIGH);
121
122     Temperature_Sensor temperature_sensor(2);
123     PH_Sensor ph_sensor(1);
124     Sensor offset_sensor(2);
125
126     temperature_sensor.set_slope(slope_temperature);
127     temperature_sensor.set_offset(offset_temperature);
128     ph_sensor.set_slope(slope_ph);
129     ph_sensor.set_offset(offset_ph);
130
131     int i= 0;
132     for(i=0; i<100; i++)
133     {
134         temperature_disable.setValue(LOW);
135         float ph = ph_sensor.get_sensor_value(offset_sensor);
136         float temperature = temperature_sensor.get_sensor_value();
137         temperature_disable.setValue(HIGH);
138         cout << "The voltage value is: " << temperature << " V." << endl;
139         temperature_sensor.post_temperature_data(temperature);
140         ph_sensor.post_ph_data(ph);
141         usleep(100000);
142     }
143     float temperature = temperature_sensor.get_voltage_value();
144     cout << "The voltage value is: " << temperature << " V." << endl;
145     return 0;
146 }
```

Listing A.6: compute_beating.cpp

```

1 #include<iostream>
3 #include<fstream>
# include<string>
5 #include <curl/curl.h>
# include <sys/stat.h>
7 #include <fcntl.h>
# include<sstream>
```

```

9 #include<opencv2/opencv.hpp>      // C++ OpenCV include file
10 using namespace std;
11 using namespace cv;                // using the cv namespace too
12
13 int main()
14 {
15     ifstream point_file;
16     point_file.open("video_data.dat");
17     VideoCapture capture; // capturing from file
18     capture.open("output.mp4");
19
20     // set any properties in the VideoCapture object
21     capture.set(CV_CAP_PROP_FRAME_WIDTH,320);    // width pixels
22     capture.set(CV_CAP_PROP_FRAME_HEIGHT,240);    // height pixels
23
24     if(!capture.isOpened())
25     {
26         cout << "Capture not open." << endl;
27     }
28
29     Mat frame, firstFrame, diffFrame;
30
31     capture >> frame; //save the first frame
32     firstFrame = frame;
33     Scalar color_information = mean(firstFrame);
34     float initial_point = (color_information[0] +color_information[1]
35                             +color_information[2])/3;
36
37     int i;
38     for(i=1; i<capture.get(CV_CAP_PROP_FRAME_COUNT);i++)
39     {
40         capture >> frame;           // capture the image to the frame
41         if(frame.empty())
42         {
43             cout << "Failed to capture an image" << endl;
44             return -1;
45         }
46         absdiff(frame, firstFrame, diffFrame);
47         color_information = mean(frame);
48         float red = color_information[0];
49         float green = color_information[1];
50         float blue = color_information[2];
51
52         float mean_color = (red+green+blue)/3;
53
54         //qui poi chiamo la funzione che mi posta il punto nel db
55         float temp = (i-1);
56         temp *= 0.033;
57

```

```

    point_file << temp << '\t';
59      point_file << mean_color << '\n';

61  }
63
63  point_file.close();
65  return 0;
}

```

Listing A.7: Pins Setting

```

/*
2 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
*
4 * This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Purpose License Version 2 as
6 * published by the Free Software Foundation
*
8 * Original from: github.com/jadonk/validation-scripts/blob/master/test-capemgr/
*
10 * Modified by Fabio Busignani fbusigna@mit.edu
*
12 */
13
14 /dts-v1/;
15 /plugin/;
16
17 /{
18     compatible = "ti,beaglebone", "ti,beaglebone-black";
19     part-number = "KLab";
20     version = "00AO";
21
22     fragment@00 {
23         target = <&am33xx_pinmux>;
24
25         __overlay__ {
26             ebb_example: KLab {
27                 pinctrl-single,pins = <
28                     0x04c 0x07 // P9_16 - T - Output Mode7 pulldown
29                     0x024 0x07 // P8_13 - EV1 - Output Mode7 pulldown
30                     0x028 0x07 // P8_14 - EV4 - Output Mode7 pulldown
31                     0x03c 0x07 // P8_15 - EV3 - Output Mode7 pulldown
32                     0x038 0x07 // P8_16 - EV6 - Output Mode7 pulldown
33                     0x02c 0x07 // P8_17 - EV5 - Output Mode7 pulldown
34                     0x08c 0x07 // P8_18 - EV8 - Output Mode7 pulldown
35                     0x020 0x07 // P8_19 - EV7 - Output Mode7 pulldown
36                     0x030 0x07 // P8_44 - EV2 -Output Mode7 pulldown
37                     0x0a0 0x07 // P8_46 - PowerSupply
38                 >;
39             };
40         };
41     };
42 }

```

```

38
39         >;
40     };
41     };
42   };

44   fragment@1 {
45     target = <&ocp>;
46     __overlay__ {
47       gpio_helper {
48         compatible = "gpio-of-helper";
49         status = "okay";
50         pinctrl-names = "default";
51         pinctrl-0 = <&eklab>;
52       };
53     };
54   };
55 }
```

Listing A.8: HTTP.h

```

1 #ifndef HTTP
2 #define HTTP
3 #include<iostream>
4 #include<fstream>
5 #include<unistd.h>
6 #include<string>
7 #include<sstream>
8 #include<stdlib.h>
9 #include<curl/curl.h>

11 std::string const URL_POST = "http://CENSURED/sensor_values";
12 std::string const POST_DATA_TEMPERATURE = "sensor=temperature&value=";
13 std::string const POST_DATA_PH = "sensor=pH&value=";
14 std::string const POST_DATA_BEATING = "sensor=beating&value=";

17 void post_data_sensor(int type, float value);

19#endif // HTTP
```

Listing A.9: HTTP.cpp

```

1 #include "http.h"
2
3 void post_data_sensor(int type, float value){
4   CURL *curl;
5   CURLcode res;
6   /* In windows, this will init the winsock stuff */
```

```

1    curl_global_init(CURL_GLOBAL_ALL);
2
3    /* get a curl handle */
4    curl = curl_easy_init();
5    if(curl) {
6
7        /* First set the URL that is about to receive our POST. This URL can
8           just as well be a https:// URL if that is what should receive the
9           data. */
10
11        curl_easy_setopt(curl, CURLOPT_URL, URL_POST.c_str());
12
13        /* Now specify the POST data */
14
15        std::ostringstream ss;
16
17        ss << value;
18
19        std::string s;
20
21        if ( type == 0)
22            s= POST_DATA_TEMPERATURE +(ss.str());
23        else if (type == 1)
24            s= POST_DATA_PH +(ss.str());
25        else
26            s= POST_DATA_BEATING +(ss.str());
27
28        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, s.c_str());
29
30        /* Perform the request, res will get the return code */
31
32        res = curl_easy_perform(curl);
33
34        /* Check for errors */
35
36        if(res != CURLE_OK)
37            fprintf(stderr, "curl_easy_perform() failed: %s\n",
38                    curl_easy_strerror(res));
39
40        /* always cleanup */
41
42        curl_easy_cleanup(curl);
43    }
44
45    curl_global_cleanup();
46}

```

A.2 VIDEO STORING SOFTWARE

Listing A.10: main.cpp

```

1 #include <QCoreApplication>
2
3 #include "downloader.h"
4
5 #include "mytimer.h"
6
7 #include <QDebug>
8
9 int main(int argc, char *argv[])
10 {
11
12     QCoreApplication a(argc, argv);
13
14     MyTimer t;
15
16     return a.exec();
17 }

```

```
11 }
```

Listing A.11: VideoStoring.pro

```

1 #-----
2 #
3 # Project created by QtCreator 2015-02-17T21:56:37
4 #
5 #-----
6
7 QT      += core
8 QT      += network
9
10 QT      -= gui
11
12 #RC_FILE = myapp.rc
13
14 TARGET = VideoStoring
15 CONFIG  += console
16 CONFIG  -= app_bundle
17
18 TEMPLATE = app
19
20
21 SOURCES += main.cpp \
22             downloader.cpp \
23             mytimer.cpp
24
25 HEADERS += \
26             downloader.h \
27             mytimer.h

```

Listing A.12: mytimer.h

```

1 #ifndef MYTIMER_H
2 #define MYTIMER_H
3
4 #include <QObject>
5 #include <QtCore>
6 #include <downloader.h>
7
8 class MyTimer : public QObject
9 {
10     Q_OBJECT
11 public:
12     MyTimer();
13     ~MyTimer();
14     QTimer *timer;
15 }

```

```

17 public slots:
18     void timerSlot();
19 private:
20     Downloader d;
21 };
22
23 #endif // MYTIMER_H

```

Listing A.13: mytimer.cpp

```

1 #include "mytimer.h"
2 #include <QtCore>
3 #include <QDebug>
4 MyTimer::MyTimer()
5 {
6     timer = new QTimer(this);
7     connect(timer, SIGNAL(timeout()), this, SLOT(timerSlot()));
8
9     timer->start(10000);
10 }
11
12 MyTimer::~MyTimer()
13 {
14 }
15
16 void MyTimer::timerSlot(){
17     d.checkFlag();
18     if( d.getStatus() )
19     {
20         d.resetFlag();
21         d.doDownload();
22     }
23 }
24
25 }

```

Listing A.14: downloader.h

```

1 #ifndef DOWNLOADER_H
2 #define DOWNLOADER_H
3
4 #include <QObject>
5 #include <QNetworkAccessManager>
6 #include <QNetworkRequest>
7 #include <QNetworkReply>
8 #include <QUrl>
9 #include <QDateTime>
10 #include <QFile>

```

```

1 #include <QDebug>
12 #include <QEventLoop>

14 class Downloader : public QObject
{
16     Q_OBJECT
18     public:
19         explicit Downloader(QObject *parent = 0);

20         void doDownload();
21         void checkFlag();
22         bool getStatus();
23         void resetFlag();

24     signals:
25
26     public slots:
27         void replyFinished (QNetworkReply *reply);
28         void checkStatus(QNetworkReply *replay);

29     private:
30         QNetworkAccessManager *manager;
31         QNetworkAccessManager *manager2;
32         QDateTime data;
33         QString name;
34         bool flag_ready;
35         bool status;

36     };
37
38
39
40 };
41
42 #endif // DOWNLOADER_H

```

Listing A.15: downloader.cpp

```

1 #include "downloader.h"
2 const QString FLAG_ON = " True ";

4 Downloader::Downloader(QObject *parent) :
5     QObject(parent)
6 {
7 }

8     void Downloader::doDownload()
9 {
10     manager = new QNetworkAccessManager(this);
11
12     connect(manager, SIGNAL(finished(QNetworkReply*)),
13             this, SLOT(replyFinished(QNetworkReply*)));
14

```

```
16     manager->get(QNetworkRequest(
17         QUrl("http://CENSURED/video/view")));
18 }
19
20 void Downloader::replyFinished (QNetworkReply *reply)
21 {
22     if(reply->error())
23     {
24         qDebug() << "ERROR!";
25         qDebug() << reply->errorString();
26     }
27     else
28     {
29         qDebug() << reply->header(QNetworkRequest::ContentTypeHeader).toString();
30         qDebug() << reply->header(QNetworkRequest::LastModifiedHeader).
31             toString();
32         qDebug() << reply->header(QNetworkRequest::ContentLengthHeader).
33             toULongLong();
34         qDebug() << reply->attribute(QNetworkRequest::HttpStatusCodeAttribute).
35            ToInt();
36         qDebug() << reply->attribute(QNetworkRequest::HttpReasonPhraseAttribute).
37             toString();
38
39         data = QDateTime::currentDateTime();
40         name = "C:/Video/" + data.toString("yyyy-MM-dd-HH-mmss") + ".mpeg";
41
42         QFile *file = new QFile(name);
43         if(file->open(QFile::Append))
44         {
45             file->write(reply->readAll());
46             file->flush();
47             file->close();
48         }
49         delete file;
50     }
51
52     reply->deleteLater();
53 }
54
55
56 void Downloader::checkStatus(QNetworkReply *replay)
57 {
58     QString result(replay->readAll());
59     int compare = QString::compare(result, FLAG_ON);
60     if(compare == 0)
61     {
62         status = true;
63     }
64 }
```

```
64     else
65     {
66         status = false;
67     }
68     qDebug() << status;
69     flag_ready = true;
70 }
71 void Downloader::checkFlag()
72 {
73     manager = new QNetworkAccessManager(this);
74     QNetworkReply *reply = manager->get(QNetworkRequest
75             (QUrl("http://CENSURED/electrovalves/video")));
76     QEventLoop loop;
77     connect(reply, SIGNAL(finished()), &loop, SLOT(quit()));

78     loop.exec();
79     qDebug() << "I'm looking for a new video";
80     QString result(reply->readAll());
81     int compare = QString::compare(result, FLAG_ON);
82     if(compare == 0)
83     {
84         status = true;
85         qDebug() << "Found it";
86     }
87     else
88     {
89         status = false;
90         qDebug() << "No new video available";
91     }
92 }
93
94 }
95
96 bool Downloader::getStatus()
97 {
98     return status;
99 }
100

101 void Downloader::resetFlag()
102 {
103     status = false;
104     manager2 = new QNetworkAccessManager(this);
105
106     QUrl flag_url = QUrl("http://CENSURED/add/electrovalve");
107     QByteArray postData;
108     postData.append("name=video&status=off");
109     manager2->post(QNetworkRequest(flag_url), postData);
110 }
```

A.3 GOOGLE APP ENGINE

Listing A.16: Main Script of Server

```

1 from flask import Flask

3 app = Flask(__name__)

5 from flask import request
6 from flask import make_response
7
8 from models import MESSAGES
9 from google.appengine.ext import ndb
10 from google.appengine.api import memcache
11 import logging
12 import datetime
13 import cloudstorage
14 import json
15
16 SECONDS_BETWEEN_UPDATES = 60
17 SENSOR_TIMEOUT_IN_SECONDS = 300
18 ELECTROVALVES_TIME_OUT = 1000
19 BUCKET_NAME = "/CENSURED/"
20 ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'mp4'}
21 UPDATE_FLAG = 'database_updated_recently'

22 #DataPoint contains all the points that have to be plotted
23 class DataPoint(ndb.Model):
24     #the y-axes value
25     value = ndb.FloatProperty()
26     #the x-axes value (autofilled with the current data)
27     date = ndb.DateTimeProperty(auto_now_add=True)
28
29     @classmethod
30         # returns the list of DataPoint associated with sensor_key
31     def points_for_sensor(cls, sensor_key):
32         return cls.query(ancestor=sensor_key).order(-cls.date)

33     @classmethod
34         #returns the last DataPoint inserted
35     def oldest_points(cls):
36         return cls.query().order(+cls.date)

37 #Sensor class contains the list of used sensors
38 class Sensor(ndb.Model):
39     @classmethod
40         # returns that list
41     def sensor_list(cls):
42
43

```

```
45     return cls.query()

47

48 @app.route('/update')
49     def update():
50         # Check to see if we have updated recently
51         update_flag = memcache.get(UPDATE_FLAG)
52         if update_flag is not None:
53             return
54
55         memcache.add(UPDATE_FLAG, 'YES', SECONDS_BETWEEN_UPDATES)
56
57         # Delete old data points
58
59         now = datetime.datetime.now()
60         count = 0
61
62         for point in DataPoint.oldest_points().iter():
63             delta = (now - point.date).total_seconds()
64             if delta > 3600:
65                 point.key.delete()
66                 count += 1
67             else:
68                 break
69
70         logging.info("Deleted {} old data points".format(count))
71
72         # Loop through all of the sensors
73         sensor_names = []
74
75         for sensor in Sensor.sensor_list().iter():
76             # Store current values of each sensor
77             sensor_value = memcache.get(sensor.key.id() + "_value")
78             if sensor_value is not None:
79                 point = DataPoint(parent=sensor.key, value=sensor_value)
80                 point.put()
81
82             # Delete sensors that don't have any data
83             curr_points = DataPoint.points_for_sensor(sensor.key)
84             if curr_points.count() == 0 and sensor_value is None:
85                 logging.info("Deleting sensor {}".format(sensor.key.id()))
86                 sensor.key.delete()
87             else:
88                 sensor_names.append(sensor.key.id())
89
90         # Store a list of sensor names in memcache
91         memcache.set("sensor_names", ",".join(sensor_names))

92
93 # this method is used to obtain the json format containg the list of the
94 # entire sensor list made by {value, time, sensor_name} when the http
95 # method is GET or to load a new value for a given sensor using the POST
96 # http method. In this last case the data to be passed with the POST are
97 # "sensor", which indicates the sensor name, and "value", which is the
98 # float number of the sensor's value.
99
100 @app.route('/', methods=['POST'])
101 @app.route('/sensor_values', methods=['GET', 'POST'])
102
103 def process_values():
104     if request.method == 'GET':
105         sensor_names = memcache.get('sensor_names')
```

```

    if sensor_names is None:
        sensor_names = ",".join([sensor.key.id() for sensor in
            Sensor.sensor_list()])
        memcache.set('sensor_names', sensor_names)
        sensor_names = sensor_names.split(",")
        sensor_values = {}
        for sensor_name in sensor_names:
            sensor_value = memcache.get(sensor_name + '_value')
            if sensor_value is not None:
                sensor_values[sensor_name] = sensor_value
        return json.dumps(sensor_values)
    elif request.method == 'POST':
        update()
        logging.info(request.values)
        sensor_name = request.form.get('sensor')
        sensor_value = float(request.form.get('value'))
        sensor_names = memcache.get('sensor_names')
        if sensor_names is not None and sensor_name not in sensor_names:
            sensor = Sensor()
            sensor.key = ndb.Key('Sensor', sensor_name)
            sensor.put()
            sensor_names = sensor_name if sensor_names == "" else sensor_names +
                "," + sensor_name
            memcache.set('sensor_names', sensor_names)
            memcache.set(sensor_name + '_value', sensor_value,
                        SENSOR_TIMEOUT_IN_SECONDS)
        return "Success\n"
    # this function is accessible only using a http GET method, it returns a
    # json object with the sensor's names list
    @app.route('/sensor_names', methods=['GET'])
    def print_names():
        return json.dumps([sensor.key.id() for sensor in Sensor.sensor_list()])
    # this function is accessible only through GET http method. This GET request
    # has to contain two parameters one is "sensor", the sensor name, and the
    # other one is "first_timestamp", the timestamp of the first point that has
    # to be plotted (this last parameter is not mandatory, if missed it is assumed
    # equal to 0.
    @app.route('/graphing_data', methods=['GET'])
    def graphing_data():
        first_timestamp = request.args.get('first_timestamp')
        sensor_name = request.args.get('sensor')
        if first_timestamp is None:
            first_timestamp = 0
        else:
            first_timestamp = float(first_timestamp)
        points = DataPoint.points_for_sensor(ndb.Key('Sensor', sensor_name))
        epoch = datetime.datetime(1970, 1, 1)

```

```

143     res = []
144     for point in points.iter():
145         timestamp = (point.date - epoch).total_seconds()
146         if timestamp > first_timestamp:
147             _res = {"timestamp": timestamp, "value": point.value}
148             res.append(_res)
149         else:
150             break
151     res.reverse()
152     return json.dumps(res)
153
154 # this function is accessible through GET http method and deletes all the
155 # point of sensors
156 @app.route('/clear', methods=['GET'])
157 def clear_data():
158     points = DataPoint.query()
159     for point in points:
160         point.key.delete()
161     for sensor in Sensor.sensor_list():
162         sensor.key.delete()
163     return "Success"
164
165 # this function is accessible through both GET and POST http methods and it
166 # is in charge to store the picture of the beating plot. This function is
167 # normally used with the POST method where the image is passed through
168 # "Image.jpg" field. Using a PC is also possible to update an image
169 # just using a browser, thank to the GET method implementation
170 @app.route('/picture/submit', methods=['GET', 'POST'])
171 def set_picture():
172     if request.method == 'POST':
173
174         _file = request.files['Image.jpg']
175
176         if _file:
177             cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_image."
178                                         + _file.filename.rsplit('.', 1)[1],
179                                         mode='w', content_type="image/jpeg")
180             _file.save(cloud_file)
181             cloud_file.close()
182             return "Success"
183     else:
184         return ''
185     <!doctype html>
186     <title>Upload microscope file V.1</title>
187     <h1>Upload microscope file</h1>
188     <form method="POST">
189         action=""
190         role="form"
191         enctype="multipart/form-data">

```

```
193     <p><input type=file name=Image.jpg>
194         <input type=submit value=Upload>
195     </form>
196     ,,
197
# this function is accessible through a GET http method and it returns the image
199 # stored inside the datastore
200 @app.route('/picture/view', methods=['GET'])
201 def view_picture():
202     for _file in cloudstorage.listbucket(BUCKET_NAME):
203         if "microscope_image" in _file.filename:
204             _file = cloudstorage.stat(_file.filename)
205             logging.info(_file.filename)
206             logging.info(_file.content_type)
207             cloud_file = cloudstorage.open(_file.filename, mode='r')
208             response = make_response(cloud_file.read())
209             cloud_file.close()
210             response.mimetype = _file.content_type
211             return response
212     return "No file found"
213
# this function is accessible through both GET and POST http methods and it
214 # is in charge to store the microscope video. This function is normally used
215 # with the POST method where the image is passed through "Video.mp4" field.
216 # Using a PC is also possible to update a video just using a browser,
217 # thank to the GET method implementation
218 @app.route('/video/submit', methods=['GET', 'POST'])
219 def set_video():
220     if request.method == 'GET':
221         return ''
222         <!doctype html>
223         <title>Upload microscope video</title>
224         <h1>Upload microscope video</h1>
225         <form method="POST">
226             action=""
227             role="form"
228             enctype="multipart/form-data">
229             <p><input type=file name=Video.mp4>
230                 <input type=submit value=Upload>
231             </form>
232             ,,
233
234     else:
235         _file = request.files['Video.mp4']
236         if _file:
237             cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_video." +
238                                         _file.filename.rsplit('.', 1)[1],
239                                         mode='w', content_type="video/mpeg")
240             _file.save(cloud_file)
```

```

241         cloud_file.close()
242         return "Success"
243
244
245 # this function is accessible through a GET http method and it returns the
# microscope video stored inside the datastore
246 @app.route('/video/view', methods=['GET'])
247 def view_video():
248     for _file in cloudstorage.listbucket(BUCKET_NAME):
249         if "microscope_video" in _file.filename:
250             _file = cloudstorage.stat(_file.filename)
251             logging.info(_file.filename)
252             logging.info(_file.content_type)
253             cloud_file = cloudstorage.open(_file.filename, mode='r')
254             response = make_response(cloud_file.read())
255             cloud_file.close()
256             response.mimetype = _file.content_type
257             return response
258
259     return "No file found"
260
261 # this function is accessible through a POST http method and it is in
# charge to store the electrovalve status inside the memcache. This http
262 # POST has to have the following parameters: "name", the name that
# identifies the valve, "status", the status of electrovalve (on, off)
263 @app.route('/add/electrovalve', methods=['POST'])
264 def add_electrovalve():
265     key = request.form.get('name')
266     message = request.form.get('status')
267     if key and message:
268         if message == 'on' or message == 'On' or message == 'ON':
269             MESSAGES[key] = True
270             memcache.set(key, True)
271
272         else:
273             MESSAGES[key] = False
274             memcache.set(key, False)
275
276         # ev.put()
277
278     return '%r' % memcache.get(key)
279
280
281 # this function accessible through a GET method returns the value of the
# valve identified through <name> field in the URL
282 @app.route('/electrovalves/<name>', methods=['GET'])
283 def get_electrovalve(name):
284     ev = memcache.get(name);
285     if ev is None:
286         return ' %r ' % MESSAGES[name] or "%s not found!" % name
287     else:
288         return ' %r ' % memcache.get(name)
289

```

```
291 @app.errorhandler(404)
292     def page_not_found(e):
293         """Return a custom 404 error."""
294         return 'Sorry, nothing at this URL.', 404
295
296
297 if __name__ == '__main__':
298     app.run()
```

A.4 GLASSWARE

Listing A.17: MainService.java

```
1 package com.google.android.glass.sample.klabinterface;
2
3 import com.google.android.glass.timeline.LiveCard;
4 import com.google.android.glass.timeline.LiveCard.PublishMode;
5
6 import android.app.PendingIntent;
7 import android.app.Service;
8 import android.content.Context;
9 import android.content.Intent;
10 import android.graphics.Bitmap;
11 import android.graphics.BitmapFactory;
12 import android.net.ConnectivityManager;
13 import android.net.NetworkInfo;
14 import android.os.AsyncTask;
15 import android.os.Environment;
16 import android.os.Handler;
17 import android.os.HandlerThread;
18 import android.os.IBinder;
19 import android.util.Log;
20 import android.view.animation.Animation;
21
22 import com.google.android.glass.timeline.LiveCard;
23 import com.googlecode.charts4j.AxisLabelsFactory;
24 import com.googlecode.charts4j.Data;
25 import com.googlecode.charts4j.GCharts;
26 import com.googlecode.charts4j.LineChart;
27 import com.googlecode.charts4j.Plot;
28 import com.googlecode.charts4j.Plots;
29 import com.jjoe64.graphview.GraphView;
30 import com.jjoe64.graphview.GraphViewSeries;
31
32 import org.json.JSONArray;
33 import org.json.JSONException;
34 import org.json.JSONObject;
35 import org.json.JSONTokener;
36
37 import com.jjoe64.graphview.GraphView;
38 import com.jjoe64.graphview.LineGraphView;
39
40 import java.io.BufferedInputStream;
41 import java.io.File;
42 import java.io.FileOutputStream;
43 import java.io.IOException;
```

```
import java.io.InputStream;
46 import java.net.HttpURLConnection;
    import java.net.URL;
48 import java.net.URLConnection;
    import java.nio.channels.FileLock;
50 import java.util.ArrayList;
    import java.util.Collections;
52 import java.util.HashMap;
    import java.util.Map;

54

56 /**
 * Service owning the LiveCard living in the timeline.
 */
58

public class MainService extends Service {
    /** TAG associated to the LiveCard */
    private static final String LIVE_CARD_TAG = "BWH interface";
    /** TAG associated to the Menu view */
    private static final String MENU_TAG = "Menu";
    /** TAG associated to the Temperature view */
    private static final String TEMPERATURE_TAG = "Temperature";
    /** TAG associated to the PH view */
    private static final String PH_TAG = "pH";
    /** TAG associated to the Video view */
    private static final String VIDEO_TAG = "Video";
    /** TAG associated to the Beating view */
    private static final String BEATING_TAG = "Beating";
    /** TAG associated to the Electrovalves view */
    private static final String ELECTROVALVES_TAG = "Electrovalve";

    private Bitmap bmp;

    /** URL where the image is stored */
    private static final String URL_IMAGE =
        "http://CENSURED/picture/view";

    /** Runnable which describes the task to download the Beating's graph */
    private final ImageDownloader mImageDownloader =
        new ImageDownloader(URL_IMAGE, this);
    /** Action is an enumerate used to implement switch-case for the extra
     * text appended to the intent */
    private static enum Action
    {
        Menu, Temperature, pH, Video, Beating, Electrovalves
    }

    private AppDrawer mCallback;
```

```

94     private LiveCard mLiveCard;

96     /** HandlerThread used to launch a background thread that manages the
97      * Data updating */
98     private HandlerThread mHandlerThread;
99     /** Handler used to launch a background thread that manages the Data updating*/
100    private Handler mHandler;

102    /** INT associated to the Menu view request */
103    private static final int MENU = 0;
104    /** INT associated to the PH view request */
105    private static final int PH = 1;
106    /** INT associated to the Menu view request */
107    private static final int TEMPERATURE = 2;
108    /** INT associated to the Video view request */
109    private static final int VIDEO = 3;
110    /** INT associated to the Beating view request */
111    private static final int BEATING = 4;

112    /** updating data period (in ms) */
113    private static final long DATA_UPDATE_DELAY_MILLIS = 500;
114    /** updating graph period (in ms) */
115    private static final long GRAPH_UPDATE_DELAY_MILLIS = 500;
116    /** updating video period (in ms) */
117    private static final long FRAME_TIME_MILLIS = 100;
118    private static final long VIDEO_UPDATE_DELAY_MILLIS = 60*1000; //time for video

119

120    /** Runnable which describes the task to update the pH and Temperature
121     * sensors values */
122    private final UpdateSensorValuesRunnable mUpdateSensorValuesRunnable =
123                                new UpdateSensorValuesRunnable();
124

125    /** Runnable which describes the task to compute the pH and Temperature
126     * graph (starting from their value) */
127    private final UpdateSensorGraphsRunnable mUpdateSensorGraphsRunnable =
128                                new UpdateSensorGraphsRunnable();

129

130    private final UpdateMicroscopeVideoRunnable mUpdateMicroscopeVideoRunnable =
131                                new UpdateMicroscopeVideoRunnable();
132

133    /** Runnable which describes the task to update the Electrovalves status */
134    private final UpdateElectrovalvesStatus mUpdateElectrovalvesStatus =
135                                new UpdateElectrovalvesStatus();

136

137

138    private static final String VIDEO_FILE_NAME =
139        Environment.getExternalStorageDirectory() + "/microscope_video.mp4";
140    private static final String TEMP_VIDEO_FILE_NAME =
141        Environment.getExternalStorageDirectory() + "/temp_microscope_video.mp4";
142

```

```
144     /** String array for the sensors (PH and Temperature) */
145     private String[] mSensors = new String[]{PH_TAG, TEMPERATURE_TAG};
146
147     //          Hash table used for creating the graphs
148     /** Hash table in which the sensors values are stored */
149     private Map<String,Double> mCurrentSensorValues;
150
151     /** Hash table in which the graphs points are stored */
152     private Map<String, ArrayList<DataPoint>> mSensorGraphData;
153
154     /** Hash table in which the sensors graphs are stored */
155     private Map<String,Bitmap> mCurrentSensorGraphs;
156
157
158     private Map<String,Double> mSensorAverage;
159
160
161     @Override
162     public IBinder onBind(Intent intent) {
163         return null;
164     }
165
166
167     @Override
168     public int onStartCommand(Intent intent, int flags, int startId) {
169         if (mLiveCard == null) {
170             AppManager.getInstance().setState(MENU);
171             mLiveCard = new LiveCard(this, LIVE_CARD_TAG);
172
173             // Keep track of the callback to remove it before unpublishing.
174             mCallback = new AppDrawer(this);
175             mLiveCard.setDirectRenderingEnabled(true).getSurfaceHolder()
176                         .addCallback(mCallback);
177
178
179             Intent menuIntent = new Intent(this, MenuActivity.class);
180             menuIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
181                                 Intent.FLAG_ACTIVITY_CLEAR_TASK);
182             mLiveCard.setAction(PendingIntent.getActivity(this, 0, menuIntent, 0));
183             mLiveCard.attach(this);
184             mLiveCard.publish(PublishMode.REVEAL);
185
186
187             /* Launch the task to update the data in another thread */
188             mHandlerThread = new HandlerThread("myHandlerThread");
189             mHandlerThread.start();
190             mHandler = new Handler(mHandlerThread.getLooper());
191             DataTask task = new DataTask();
192             task.execute(this);
193         } else {
194             if(intent.getStringExtra(Intent.EXTRA_TEXT)!= null) {
195                 Action action = Action.valueOf(intent.getStringExtra
196                                         (Intent.EXTRA_TEXT));
197                 switch (action) {
198                     case Menu:
```

```

192             Log.i(LIVE_CARD_TAG, "State = Menu");
193             AppManager.getInstance().setState(MENU);
194             break;
195
196         case Beating:
197             Log.i(LIVE_CARD_TAG, "State = Beating");
198             AppManager.getInstance().setState(BEATING);
199             break;
200
201         case pH:
202             Log.i(LIVE_CARD_TAG, "State = pH");
203             AppManager.getInstance().setState(PH);
204             break;
205
206         case Temperature:
207             Log.i(LIVE_CARD_TAG, "State = Temperature");
208             AppManager.getInstance().setState(TEMPERATURE);
209             break;
210
211         case Video:
212             Log.i(LIVE_CARD_TAG, "State = Video");
213             AppManager.getInstance().setState(VIDEO);
214             break;
215
216         case Electrovalves:
217             Log.i(LIVE_CARD_TAG, "State = Electrovalves");
218             AppManager.getInstance().setState(ELECTROVALVES);
219             break;
220
221         default:
222             mLiveCard.navigate();
223             break;
224         }
225     }
226
227     else
228     {
229         mLiveCard.navigate();
230     }
231 }
232
233
234     @Override
235     public void onDestroy() {
236
237         // Stop the Task which update the beating image
238         if(!mImageDownloader.isStopped())
239         {
240
241             mImageDownloader.setIsStopped(true);
242             Log.i(BEATING_TAG, "Removed Task");
243         }
244     }

```

```

    // Stop the Task which update the sensors Graphs
242    if (!mUpdateSensorGraphsRunnable.isStopped())
    {
244        mUpdateSensorGraphsRunnable.setStop(true);
        Log.i(PH_TAG + " " + TEMPERATURE_TAG, "Removed Task of Graphs");
    }
    // Stop the Task which update the sensors value
248    if (!mUpdateSensorValuesRunnable.isStopped())
    {
250        mUpdateSensorValuesRunnable.setStop(true);
        Log.i(PH_TAG + " " + TEMPERATURE_TAG, "Removed Task of Values");
    }
    if (!mUpdateMicroscopeVideoRunnable.isStopped())
    {
254        mUpdateMicroscopeVideoRunnable.setStop(true);
        Log.i(VIDEO_TAG, "Removed Task of Uploading values");
    }
258    if (!mUpdateElectrovalvesStatus.isStopped())
    {
260        mUpdateElectrovalvesStatus.setStop(true);
        Log.i(ELECTROVALVES_TAG, "Removed Task for getting
262                           Electrovalves status");
    }
264    if (mLiveCard != null && mLiveCard.isPublished()) {
        mLiveCard.unpublish();
        mLiveCard = null;
    }
268    super.onDestroy();
}

270 /**
 * Asynchronous Task for downloading the graphs and video in background
 */
274 private class DataTask extends AsyncTask<MainService,Void,Void>
{
276
    @Override
278    protected Void doInBackground(MainService... params) {

280
        Log.i(LIVE_CARD_TAG, "Loading initial data");

282
        // create the three hash tables
        mCurrentSensorValues = new HashMap<String, Double>();
        mSensorGraphData = new HashMap<String, ArrayList<DataPoint>>();
        mCurrentSensorGraphs = new HashMap<String, Bitmap>();
        mSensorAverage = new HashMap<String, Double>();

286
        // initializes each hash map with dummy values
        for (String mSensor : mSensors)

```

```

290     {
291         mCurrentSensorValues.put(mSensor, 0.0);
292         mSensorAverage.put(mSensor, 0.0);
293         mSensorGraphData.put(mSensor, new ArrayList<DataPoint>());
294         mCurrentSensorGraphs.put(mSensor, null);
295     }
296
297     Log.i(LIVE_CARD_TAG, "Download the data");
298
299     mUpdateSensorValuesRunnable.run();
300     // give the hash table to the Callback
301     // mCallback.setSensorValues(mCurrentSensorValues);
302     // compute the graphs with previous values and give them to
303     // the AppDrawer
304     mUpdateSensorGraphsRunnable.run();
305
306     // give the hash table to the Callback
307     // mCallback.setSensorGraphs(mCurrentSensorGraphs);
308     // download the beating image and give this to the AppDrawer
309     mImageDownloader.run();
310
311     mUpdateMicroscopeVideoRunnable.run();
312
313     mUpdateElectrovalvesStatus.run();
314     return null;
315 }
316
317 /**
318 * This runnable updates the sensors values taking them from the google engine
319 */
320 private class UpdateSensorValuesRunnable implements Runnable
321 {
322     private boolean mIsStopped = false;
323
324     /** It implements the task of runnable
325      *
326      * @see UpdateSensorValuesRunnable
327      */
328
329     @Override
330     public void run()
331     {
332         if (!mIsStopped)
333         {
334             /** JavaScript object, where the sensor values are
335              * temporary stored */
336             JSONObject values = getSensorValues();
337             if (values != null)
338             {

```

```

340             for (String mSensor : mSensors)
341             {
342                 try
343                 {
344                     // update the hash table of sensor values
345                     mCurrentSensorValues.put(mSensor,
346                         values.getDouble(mSensor));
347                 }
348             catch (JSONException ignored)
349             {}
350         }
351     }
352
353     // restart the Runnable after a given amount of time
354     mHandler.postDelayed(mUpdateSensorValuesRunnable,
355                           DATA_UPDATE_DELAY_MILLIS);
356 }
357
358
359
360 /**
361 * It is the getter function that shows the status of runnable
362 *
363 * @return mIsStopped, true if the runnable is stopped, false otherwise
364 */
365 public boolean isStopped()
366 {
367     return mIsStopped;
368 }
369
370 /**
371 * It is the setter that allows to stop the runnable
372 *
373 * @param isStopped, true if the user wishes to stop the runnable,
374 * false otherwise
375 */
376 public void setStop(boolean isStopped)
377 {
378     this.mIsStopped = isStopped;
379 }
380
381 /**
382 * It gets the values of all of the sensors
383 *
384 * @return JSONObject which contains all the values of sensors
385 */
386 private JSONObject getSensorValues()
387 {
388     try
389     {
390         JSONObject jsonObject = new JSONObject();
391
392         for (String mSensor : mSensors)
393         {
394             try
395             {
396                 // update the hash table of sensor values
397                 mCurrentSensorValues.put(mSensor,
398                     values.getDouble(mSensor));
399             }
400             catch (JSONException ignored)
401             {}
402         }
403
404         // restart the Runnable after a given amount of time
405         mHandler.postDelayed(mUpdateSensorValuesRunnable,
406                               DATA_UPDATE_DELAY_MILLIS);
407     }
408
409     return jsonObject;
410 }
411
412 /**
413 * It updates the sensor values
414 */
415 public void updateSensorValues()
416 {
417     try
418     {
419         JSONObject jsonObject = new JSONObject();
420
421         for (String mSensor : mSensors)
422         {
423             try
424             {
425                 // update the hash table of sensor values
426                 mCurrentSensorValues.put(mSensor,
427                     values.getDouble(mSensor));
428             }
429             catch (JSONException ignored)
430             {}
431         }
432
433         // restart the Runnable after a given amount of time
434         mHandler.postDelayed(mUpdateSensorValuesRunnable,
435                               DATA_UPDATE_DELAY_MILLIS);
436     }
437
438     return;
439 }
440
441 /**
442 * It stops the runnable
443 */
444 public void stopRunnable()
445 {
446     mIsStopped = true;
447 }
448
449 /**
450 * It starts the runnable
451 */
452 public void startRunnable()
453 {
454     mIsStopped = false;
455 }
456
457 /**
458 * It checks if the runnable is stopped
459 */
460 public boolean isRunnableStopped()
461 {
462     return mIsStopped;
463 }
464
465 /**
466 * It gets the sensor values
467 */
468 public JSONObject getSensorValues()
469 {
470     return getSensorValues();
471 }
472
473 /**
474 * It updates the sensor values
475 */
476 public void updateSensorValues()
477 {
478     updateSensorValues();
479 }
480
481 /**
482 * It stops the runnable
483 */
484 public void stopRunnable()
485 {
486     stopRunnable();
487 }
488
489 /**
490 * It starts the runnable
491 */
492 public void startRunnable()
493 {
494     startRunnable();
495 }
496
497 /**
498 * It checks if the runnable is stopped
499 */
500 public boolean isRunnableStopped()
501 {
502     return isRunnableStopped();
503 }
504
505 /**
506 * It gets the sensor values
507 */
508 public JSONObject getSensorValues()
509 {
510     return getSensorValues();
511 }
512
513 /**
514 * It updates the sensor values
515 */
516 public void updateSensorValues()
517 {
518     updateSensorValues();
519 }
520
521 /**
522 * It stops the runnable
523 */
524 public void stopRunnable()
525 {
526     stopRunnable();
527 }
528
529 /**
530 * It starts the runnable
531 */
532 public void startRunnable()
533 {
534     startRunnable();
535 }
536
537 /**
538 * It checks if the runnable is stopped
539 */
540 public boolean isRunnableStopped()
541 {
542     return isRunnableStopped();
543 }
544
545 /**
546 * It gets the sensor values
547 */
548 public JSONObject getSensorValues()
549 {
550     return getSensorValues();
551 }
552
553 /**
554 * It updates the sensor values
555 */
556 public void updateSensorValues()
557 {
558     updateSensorValues();
559 }
560
561 /**
562 * It stops the runnable
563 */
564 public void stopRunnable()
565 {
566     stopRunnable();
567 }
568
569 /**
570 * It starts the runnable
571 */
572 public void startRunnable()
573 {
574     startRunnable();
575 }
576
577 /**
578 * It checks if the runnable is stopped
579 */
580 public boolean isRunnableStopped()
581 {
582     return isRunnableStopped();
583 }
584
585 /**
586 * It gets the sensor values
587 */
588 public JSONObject getSensorValues()
589 {
590     return getSensorValues();
591 }
592
593 /**
594 * It updates the sensor values
595 */
596 public void updateSensorValues()
597 {
598     updateSensorValues();
599 }
599
600 /**
601 * It stops the runnable
602 */
603 public void stopRunnable()
604 {
605     stopRunnable();
606 }
607
608 /**
609 * It starts the runnable
610 */
611 public void startRunnable()
612 {
613     startRunnable();
614 }
615
616 /**
617 * It checks if the runnable is stopped
618 */
619 public boolean isRunnableStopped()
620 {
621     return isRunnableStopped();
622 }
623
624 /**
625 * It gets the sensor values
626 */
627 public JSONObject getSensorValues()
628 {
629     return getSensorValues();
630 }
631
632 /**
633 * It updates the sensor values
634 */
635 public void updateSensorValues()
636 {
637     updateSensorValues();
638 }
639
640 /**
641 * It stops the runnable
642 */
643 public void stopRunnable()
644 {
645     stopRunnable();
646 }
647
648 /**
649 * It starts the runnable
650 */
651 public void startRunnable()
652 {
653     startRunnable();
654 }
655
656 /**
657 * It checks if the runnable is stopped
658 */
659 public boolean isRunnableStopped()
660 {
661     return isRunnableStopped();
662 }
663
664 /**
665 * It gets the sensor values
666 */
667 public JSONObject getSensorValues()
668 {
669     return getSensorValues();
670 }
671
672 /**
673 * It updates the sensor values
674 */
675 public void updateSensorValues()
676 {
677     updateSensorValues();
678 }
679
680 /**
681 * It stops the runnable
682 */
683 public void stopRunnable()
684 {
685     stopRunnable();
686 }
687
688 /**
689 * It starts the runnable
690 */
691 public void startRunnable()
692 {
693     startRunnable();
694 }
695
696 /**
697 * It checks if the runnable is stopped
698 */
699 public boolean isRunnableStopped()
700 {
701     return isRunnableStopped();
702 }
703
704 /**
705 * It gets the sensor values
706 */
707 public JSONObject getSensorValues()
708 {
709     return getSensorValues();
710 }
711
712 /**
713 * It updates the sensor values
714 */
715 public void updateSensorValues()
716 {
717     updateSensorValues();
718 }
719
720 /**
721 * It stops the runnable
722 */
723 public void stopRunnable()
724 {
725     stopRunnable();
726 }
727
728 /**
729 * It starts the runnable
730 */
731 public void startRunnable()
732 {
733     startRunnable();
734 }
735
736 /**
737 * It checks if the runnable is stopped
738 */
739 public boolean isRunnableStopped()
740 {
741     return isRunnableStopped();
742 }
743
744 /**
745 * It gets the sensor values
746 */
747 public JSONObject getSensorValues()
748 {
749     return getSensorValues();
750 }
751
752 /**
753 * It updates the sensor values
754 */
755 public void updateSensorValues()
756 {
757     updateSensorValues();
758 }
759
760 /**
761 * It stops the runnable
762 */
763 public void stopRunnable()
764 {
765     stopRunnable();
766 }
767
768 /**
769 * It starts the runnable
770 */
771 public void startRunnable()
772 {
773     startRunnable();
774 }
775
776 /**
777 * It checks if the runnable is stopped
778 */
779 public boolean isRunnableStopped()
780 {
781     return isRunnableStopped();
782 }
783
784 /**
785 * It gets the sensor values
786 */
787 public JSONObject getSensorValues()
788 {
789     return getSensorValues();
790 }
791
792 /**
793 * It updates the sensor values
794 */
795 public void updateSensorValues()
796 {
797     updateSensorValues();
798 }
799
800 /**
801 * It stops the runnable
802 */
803 public void stopRunnable()
804 {
805     stopRunnable();
806 }
807
808 /**
809 * It starts the runnable
810 */
811 public void startRunnable()
812 {
813     startRunnable();
814 }
815
816 /**
817 * It checks if the runnable is stopped
818 */
819 public boolean isRunnableStopped()
820 {
821     return isRunnableStopped();
822 }
823
824 /**
825 * It gets the sensor values
826 */
827 public JSONObject getSensorValues()
828 {
829     return getSensorValues();
830 }
831
832 /**
833 * It updates the sensor values
834 */
835 public void updateSensorValues()
836 {
837     updateSensorValues();
838 }
839
840 /**
841 * It stops the runnable
842 */
843 public void stopRunnable()
844 {
845     stopRunnable();
846 }
847
848 /**
849 * It starts the runnable
850 */
851 public void startRunnable()
852 {
853     startRunnable();
854 }
855
856 /**
857 * It checks if the runnable is stopped
858 */
859 public boolean isRunnableStopped()
860 {
861     return isRunnableStopped();
862 }
863
864 /**
865 * It gets the sensor values
866 */
867 public JSONObject getSensorValues()
868 {
869     return getSensorValues();
870 }
871
872 /**
873 * It updates the sensor values
874 */
875 public void updateSensorValues()
876 {
877     updateSensorValues();
878 }
879
880 /**
881 * It stops the runnable
882 */
883 public void stopRunnable()
884 {
885     stopRunnable();
886 }
887
888 /**
889 * It starts the runnable
890 */
891 public void startRunnable()
892 {
893     startRunnable();
894 }
895
896 /**
897 * It checks if the runnable is stopped
898 */
899 public boolean isRunnableStopped()
900 {
901     return isRunnableStopped();
902 }
903
904 /**
905 * It gets the sensor values
906 */
907 public JSONObject getSensorValues()
908 {
909     return getSensorValues();
910 }
911
912 /**
913 * It updates the sensor values
914 */
915 public void updateSensorValues()
916 {
917     updateSensorValues();
918 }
919
920 /**
921 * It stops the runnable
922 */
923 public void stopRunnable()
924 {
925     stopRunnable();
926 }
927
928 /**
929 * It starts the runnable
930 */
931 public void startRunnable()
932 {
933     startRunnable();
934 }
935
936 /**
937 * It checks if the runnable is stopped
938 */
939 public boolean isRunnableStopped()
940 {
941     return isRunnableStopped();
942 }
943
944 /**
945 * It gets the sensor values
946 */
947 public JSONObject getSensorValues()
948 {
949     return getSensorValues();
950 }
951
952 /**
953 * It updates the sensor values
954 */
955 public void updateSensorValues()
956 {
957     updateSensorValues();
958 }
959
960 /**
961 * It stops the runnable
962 */
963 public void stopRunnable()
964 {
965     stopRunnable();
966 }
967
968 /**
969 * It starts the runnable
970 */
971 public void startRunnable()
972 {
973     startRunnable();
974 }
975
976 /**
977 * It checks if the runnable is stopped
978 */
979 public boolean isRunnableStopped()
980 {
981     return isRunnableStopped();
982 }
983
984 /**
985 * It gets the sensor values
986 */
987 public JSONObject getSensorValues()
988 {
989     return getSensorValues();
990 }
991
992 /**
993 * It updates the sensor values
994 */
995 public void updateSensorValues()
996 {
997     updateSensorValues();
998 }
999
1000 /**
1001 * It stops the runnable
1002 */
1003 public void stopRunnable()
1004 {
1005     stopRunnable();
1006 }
1007
1008 /**
1009 * It starts the runnable
1010 */
1011 public void startRunnable()
1012 {
1013     startRunnable();
1014 }
1015
1016 /**
1017 * It checks if the runnable is stopped
1018 */
1019 public boolean isRunnableStopped()
1020 {
1021     return isRunnableStopped();
1022 }
1023
1024 /**
1025 * It gets the sensor values
1026 */
1027 public JSONObject getSensorValues()
1028 {
1029     return getSensorValues();
1030 }
1031
1032 /**
1033 * It updates the sensor values
1034 */
1035 public void updateSensorValues()
1036 {
1037     updateSensorValues();
1038 }
1039
1040 /**
1041 * It stops the runnable
1042 */
1043 public void stopRunnable()
1044 {
1045     stopRunnable();
1046 }
1047
1048 /**
1049 * It starts the runnable
1050 */
1051 public void startRunnable()
1052 {
1053     startRunnable();
1054 }
1055
1056 /**
1057 * It checks if the runnable is stopped
1058 */
1059 public boolean isRunnableStopped()
1060 {
1061     return isRunnableStopped();
1062 }
1063
1064 /**
1065 * It gets the sensor values
1066 */
1067 public JSONObject getSensorValues()
1068 {
1069     return getSensorValues();
1070 }
1071
1072 /**
1073 * It updates the sensor values
1074 */
1075 public void updateSensorValues()
1076 {
1077     updateSensorValues();
1078 }
1079
1080 /**
1081 * It stops the runnable
1082 */
1083 public void stopRunnable()
1084 {
1085     stopRunnable();
1086 }
1087
1088 /**
1089 * It starts the runnable
1090 */
1091 public void startRunnable()
1092 {
1093     startRunnable();
1094 }
1095
1096 /**
1097 * It checks if the runnable is stopped
1098 */
1099 public boolean isRunnableStopped()
1100 {
1101     return isRunnableStopped();
1102 }
1103
1104 /**
1105 * It gets the sensor values
1106 */
1107 public JSONObject getSensorValues()
1108 {
1109     return getSensorValues();
1110 }
1111
1112 /**
1113 * It updates the sensor values
1114 */
1115 public void updateSensorValues()
1116 {
1117     updateSensorValues();
1118 }
1119
1120 /**
1121 * It stops the runnable
1122 */
1123 public void stopRunnable()
1124 {
1125     stopRunnable();
1126 }
1127
1128 /**
1129 * It starts the runnable
1130 */
1131 public void startRunnable()
1132 {
1133     startRunnable();
1134 }
1135
1136 /**
1137 * It checks if the runnable is stopped
1138 */
1139 public boolean isRunnableStopped()
1140 {
1141     return isRunnableStopped();
1142 }
1143
1144 /**
1145 * It gets the sensor values
1146 */
1147 public JSONObject getSensorValues()
1148 {
1149     return getSensorValues();
1150 }
1151
1152 /**
1153 * It updates the sensor values
1154 */
1155 public void updateSensorValues()
1156 {
1157     updateSensorValues();
1158 }
1159
1160 /**
1161 * It stops the runnable
1162 */
1163 public void stopRunnable()
1164 {
1165     stopRunnable();
1166 }
1167
1168 /**
1169 * It starts the runnable
1170 */
1171 public void startRunnable()
1172 {
1173     startRunnable();
1174 }
1175
1176 /**
1177 * It checks if the runnable is stopped
1178 */
1179 public boolean isRunnableStopped()
1180 {
1181     return isRunnableStopped();
1182 }
1183
1184 /**
1185 * It gets the sensor values
1186 */
1187 public JSONObject getSensorValues()
1188 {
1189     return getSensorValues();
1190 }
1191
1192 /**
1193 * It updates the sensor values
1194 */
1195 public void updateSensorValues()
1196 {
1197     updateSensorValues();
1198 }
1199
1200 /**
1201 * It stops the runnable
1202 */
1203 public void stopRunnable()
1204 {
1205     stopRunnable();
1206 }
1207
1208 /**
1209 * It starts the runnable
1210 */
1211 public void startRunnable()
1212 {
1213     startRunnable();
1214 }
1215
1216 /**
1217 * It checks if the runnable is stopped
1218 */
1219 public boolean isRunnableStopped()
1220 {
1221     return isRunnableStopped();
1222 }
1223
1224 /**
1225 * It gets the sensor values
1226 */
1227 public JSONObject getSensorValues()
1228 {
1229     return getSensorValues();
1230 }
1231
1232 /**
1233 * It updates the sensor values
1234 */
1235 public void updateSensorValues()
1236 {
1237     updateSensorValues();
1238 }
1239
1240 /**
1241 * It stops the runnable
1242 */
1243 public void stopRunnable()
1244 {
1245     stopRunnable();
1246 }
1247
1248 /**
1249 * It starts the runnable
1250 */
1251 public void startRunnable()
1252 {
1253     startRunnable();
1254 }
1255
1256 /**
1257 * It checks if the runnable is stopped
1258 */
1259 public boolean isRunnableStopped()
1260 {
1261     return isRunnableStopped();
1262 }
1263
1264 /**
1265 * It gets the sensor values
1266 */
1267 public JSONObject getSensorValues()
1268 {
1269     return getSensorValues();
1270 }
1271
1272 /**
1273 * It updates the sensor values
1274 */
1275 public void updateSensorValues()
1276 {
1277     updateSensorValues();
1278 }
1279
1280 /**
1281 * It stops the runnable
1282 */
1283 public void stopRunnable()
1284 {
1285     stopRunnable();
1286 }
1287
1288 /**
1289 * It starts the runnable
1290 */
1291 public void startRunnable()
1292 {
1293     startRunnable();
1294 }
1295
1296 /**
1297 * It checks if the runnable is stopped
1298 */
1299 public boolean isRunnableStopped()
1300 {
1301     return isRunnableStopped();
1302 }
1303
1304 /**
1305 * It gets the sensor values
1306 */
1307 public JSONObject getSensorValues()
1308 {
1309     return getSensorValues();
1310 }
1311
1312 /**
1313 * It updates the sensor values
1314 */
1315 public void updateSensorValues()
1316 {
1317     updateSensorValues();
1318 }
1319
1320 /**
1321 * It stops the runnable
1322 */
1323 public void stopRunnable()
1324 {
1325     stopRunnable();
1326 }
1327
1328 /**
1329 * It starts the runnable
1330 */
1331 public void startRunnable()
1332 {
1333     startRunnable();
1334 }
1335
1336 /**
1337 * It checks if the runnable is stopped
1338 */
1339 public boolean isRunnableStopped()
1340 {
1341     return isRunnableStopped();
1342 }
1343
1344 /**
1345 * It gets the sensor values
1346 */
1347 public JSONObject getSensorValues()
1348 {
1349     return getSensorValues();
1350 }
1351
1352 /**
1353 * It updates the sensor values
1354 */
1355 public void updateSensorValues()
1356 {
1357     updateSensorValues();
1358 }
1359
1360 /**
1361 * It stops the runnable
1362 */
1363 public void stopRunnable()
1364 {
1365     stopRunnable();
1366 }
1367
1368 /**
1369 * It starts the runnable
1370 */
1371 public void startRunnable()
1372 {
1373     startRunnable();
1374 }
1375
1376 /**
1377 * It checks if the runnable is stopped
1378 */
1379 public boolean isRunnableStopped()
1380 {
1381     return isRunnableStopped();
1382 }
1383
1384 /**
1385 * It gets the sensor values
1386 */
1387 public JSONObject getSensorValues()
1388 {
1389     return getSensorValues();
1390 }
1391
1392 /**
1393 * It updates the sensor values
1394 */
1395 public void updateSensorValues()
1396 {
1397     updateSensorValues();
1398 }
1399
1400 /**
1401 * It stops the runnable
1402 */
1403 public void stopRunnable()
1404 {
1405     stopRunnable();
1406 }
1407
1408 /**
1409 * It starts the runnable
1410 */
1411 public void startRunnable()
1412 {
1413     startRunnable();
1414 }
1415
1416 /**
1417 * It checks if the runnable is stopped
1418 */
1419 public boolean isRunnableStopped()
1420 {
1421     return isRunnableStopped();
1422 }
1423
1424 /**
1425 * It gets the sensor values
1426 */
1427 public JSONObject getSensorValues()
1428 {
1429     return getSensorValues();
1430 }
1431
1432 /**
1433 * It updates the sensor values
1434 */
1435 public void updateSensorValues()
1436 {
1437     updateSensorValues();
1438 }
1439
1440 /**
1441 * It stops the runnable
1442 */
1443 public void stopRunnable()
1444 {
1445     stopRunnable();
1446 }
1447
1448 /**
1449 * It starts the runnable
1450 */
1451 public void startRunnable()
1452 {
1453     startRunnable();
1454 }
1455
1456 /**
1457 * It checks if the runnable is stopped
1458 */
1459 public boolean isRunnableStopped()
1460 {
1461     return isRunnableStopped();
1462 }
1463
1464 /**
1465 * It gets the sensor values
1466 */
1467 public JSONObject getSensorValues()
1468 {
1469     return getSensorValues();
1470 }
1471
1472 /**
1473 * It updates the sensor values
1474 */
1475 public void updateSensorValues()
1476 {
1477     updateSensorValues();
1478 }
1479
1480 /**
1481 * It stops the runnable
1482 */
1483 public void stopRunnable()
1484 {
1485     stopRunnable();
1486 }
1487
1488 /**
1489 * It starts the runnable
1490 */
1491 public void startRunnable()
1492 {
1493     startRunnable();
1494 }
1495
1496 /**
1497 * It checks if the runnable is stopped
1498 */
1499 public boolean isRunnableStopped()
1500 {
1501     return isRunnableStopped();
1502 }
1503
1504 /**
1505 * It gets the sensor values
1506 */
1507 public JSONObject getSensorValues()
1508 {
1509     return getSensorValues();
1510 }
1511
1512 /**
1513 * It updates the sensor values
1514 */
1515 public void updateSensorValues()
1516 {
1517     updateSensorValues();
1518 }
1519
1520 /**
1521 * It stops the runnable
1522 */
1523 public void stopRunnable()
1524 {
1525     stopRunnable();
1526 }
1527
1528 /**
1529 * It starts the runnable
1530 */
1531 public void startRunnable()
1532 {
1533     startRunnable();
1534 }
1535
1536 /**
1537 * It checks if the runnable is stopped
1538 */
1539 public boolean isRunnableStopped()
1540 {
1541     return isRunnableStopped();
1542 }
1543
1544 /**
1545 * It gets the sensor values
1546 */
1547 public JSONObject getSensorValues()
1548 {
1549     return getSensorValues();
1550 }
1551
1552 /**
1553 * It updates the sensor values
1554 */
1555 public void updateSensorValues()
1556 {
1557     updateSensorValues();
1558 }
1559
1560 /**
1561 * It stops the runnable
1562 */
1563 public void stopRunnable()
1564 {
1565     stopRunnable();
1566 }
1567
1568 /**
1569 * It starts the runnable
1570 */
1571 public void startRunnable()
1572 {
1573     startRunnable();
1574 }
1575
1576 /**
1577 * It checks if the runnable is stopped
1578 */
1579 public boolean isRunnableStopped()
1580 {
1581     return isRunnableStopped();
1582 }
1583
1584 /**
1585 * It gets the sensor values
1586 */
1587 public JSONObject getSensorValues()
1588 {
1589     return getSensorValues();
1590 }
1591
1592 /**
1593 * It updates the sensor values
1594 */
1595 public void updateSensorValues()
1596 {
1597     updateSensorValues();
1598 }
1599
1600 /**
1601 * It stops the runnable
1602 */
1603 public void stopRunnable()
1604 {
1605     stopRunnable();
1606 }
1607
1608 /**
1609 * It starts the runnable
1610 */
1611 public void startRunnable()
1612 {
1613     startRunnable();
1614 }
1615
1616 /**
1617 * It checks if the runnable is stopped
1618 */
1619 public boolean isRunnableStopped()
1620 {
1621     return isRunnableStopped();
1622 }
1623
1624 /**
1625 * It gets the sensor values
1626 */
1627 public JSONObject getSensorValues()
1628 {
1629     return getSensorValues();
1630 }
1631
1632 /**
1633 * It updates the sensor values
1634 */
1635 public void updateSensorValues()
1636 {
1637     updateSensorValues();
1638 }
1639
1640 /**
1641 * It stops the runnable
1642 */
1643 public void stopRunnable()
1644 {
1645     stopRunnable();
1646 }
1647
1648 /**
1649 * It starts the runnable
1650 */
1651 public void startRunnable()
1652 {
1653     startRunnable();
1654 }
1655
1656 /**
1657 * It checks if the runnable is stopped
1658 */
1659 public boolean isRunnableStopped()
1660 {
1661     return isRunnableStopped();
1662 }
1663
1664 /**
1665 * It gets the sensor values
1666 */
1667 public JSONObject getSensorValues()
1668 {
1669     return getSensorValues();
1670 }
1671
1672 /**
1673 * It updates the sensor values
1674 */
1675 public void updateSensorValues()
1676 {
1677     updateSensorValues();
1678 }
1679
1680 /**
1681 * It stops the runnable
1682 */
1683 public void stopRunnable()
1684 {
1685     stopRunnable();
1686 }
1687
1688 /**
1689 * It starts the runnable
1690 */
1691 public void startRunnable()
1692 {
1693     startRunnable();
1694 }
1695
1696 /**
1697 * It checks if the runnable is stopped
1698 */
1699 public boolean isRunnableStopped()
1700 {
1701     return isRunnableStopped();
1702 }
1703
1704 /**
1705 * It gets the sensor values
1706 */
1707 public JSONObject getSensorValues()
1708 {
1709     return getSensorValues();
1710 }
1711
1712 /**
1713 * It updates the sensor values
1714 */
1715 public void updateSensorValues()
1716 {
1717     updateSensorValues();
1718 }
1719
1720 /**
1721 * It stops the runnable
1722 */
1723 public void stopRunnable()
1724 {
1725     stopRunnable();
1726 }
1727
1728 /**
1729 * It starts the runnable
1730 */
1731 public void startRunnable()
1732 {
1733     startRunnable();
1734 }
1735
1736 /**
1737 * It checks if the runnable is stopped
1738 */
1739 public boolean isRunnableStopped()
1740 {
1741     return isRunnableStopped();
1742 }
1743
1744 /**
1745 * It gets the sensor values
1746 */
1747 public JSONObject getSensorValues()
1748 {
1749     return getSensorValues();
1750 }
1751
1752 /**
1753 * It updates the sensor values
1754 */
1755 public void updateSensorValues()
1756 {
1757     updateSensorValues();
1758 }
1759
1760 /**
1761 * It stops the runnable
1762 */
1763 public void stopRunnable()
1764 {
1765     stopRunnable();
1766 }
1767
1768 /**
1769 * It starts the runnable
1770 */
1771 public void startRunnable()
1772 {
1773     startRunnable();
1774 }
1775
1776 /**
1777 * It checks if the runnable is stopped
1778 */
1779 public boolean isRunnableStopped()
1780 {
1781     return isRunnableStopped();
1782 }
1783
1784 /**
1785 * It gets the sensor values
1786 */
1787 public JSONObject getSensorValues()
1788 {
1789     return getSensorValues();
1790 }
1791
1792 /**
1793 * It updates the sensor values
1794 */
1795 public void updateSensorValues()
1796 {
1797     updateSensorValues();
1798 }
1799
1800 /**
1801 * It stops the runnable
1802 */
1803 public void stopRunnable()
1804 {
1805     stopRunnable();
1806 }
1807
1808 /**
1809 * It starts the runnable
1810 */
1811 public void startRunnable()
1812 {
1813     startRunnable();
1814 }
1815
1816 /**
1817 * It checks if the runnable is stopped
1818 */
1819 public boolean isRunnableStopped()
1820 {
1821     return isRunnableStopped();
1822 }
1823
1824 /**
1825 * It gets the sensor values
1826 */
1827 public JSONObject getSensorValues()
1828 {
1829     return getSensorValues();
1830 }
1831
1832 /**
1833 * It updates the sensor values
1834 */
1835 public void updateSensorValues()
1836 {
1837     updateSensorValues();
1838 }
1839
1840 /**
1841 * It stops the runnable
1842 */
1843 public void stopRunnable()
1844 {
1845     stopRunnable();
1846 }
1847
1848 /**
1849 * It starts the runnable
1850 */
1851 public void startRunnable()
1852 {
1853     startRunnable();
1854 }
1855
1856 /**
1857 * It checks if the runnable is stopped
1858 */
1859 public boolean isRunnableStopped()
1860 {
1861     return isRunnableStopped();
1862 }
1863
1864 /**
1865 * It gets the sensor values
1866 */
1867 public JSONObject getSensorValues()
1868 {
1869     return getSensorValues();
1870 }
1871
1872 /**
1873 * It updates the sensor values
1874 */
1875 public void updateSensorValues()
1876 {
1877     updateSensorValues();
1878 }
1879
1880 /**
1881 * It stops the runnable
1882 */
1883 public void stopRunnable()
1884 {
1885     stopRunnable();
1886 }
1887
1888 /**
1889 * It starts the runnable
1890 */
1891 public void startRunnable()
1892 {
1893     startRunnable();
1894 }
1895
1896 /**
1897 * It checks if the runnable is stopped
1898 */
1899 public boolean isRunnableStopped()
1900 {
1901     return isRunnableStopped();
1902 }
1903
1904 /**
1905 * It gets the sensor values
1906 */
1907 public JSONObject getSensorValues()
1908 {
1909     return getSensorValues();
1910 }
1911
1912 /**
1913 * It updates the sensor values
1914 */
1915 public void updateSensorValues()
1916 {
1917     updateSensorValues();
1918 }
1919
1920 /**
1921 * It stops the runnable
1922 */
1923 public void stopRunnable()
1924 {
1925     stopRunnable();
1926 }
1927
1928 /**
1929 * It starts the runnable
1930 */
1931 public void startRunnable()
1932 {
1933     startRunnable();
1934 }
1935
1936 /**
1937 * It checks if the runnable is stopped
1938 */
1939 public boolean isRunnableStopped()
1940 {
1941     return isRunnableStopped();
1942 }
1943
1944 /**
1945 * It gets the sensor values
1946 */
1947 public JSONObject getSensorValues()
1948 {
1949     return getSensorValues();
1950 }
1951
1952 /**
1953 * It updates the sensor values
1954 */
1955 public void updateSensorValues()
1956 {
1957     updateSensorValues();
1958 }
1959
1960 /**
1961 * It stops the runnable
1962 */
1963 public void stopRunnable()
1964 {
1965     stopRunnable();
1966 }
1967
1968 /**
1969 * It starts the runnable
1970 */
1971 public void startRunnable()
1972 {
1973     startRunnable();
1974 }
1975
1976 /**
1977 * It checks if the runnable is stopped
1978 */
1979 public boolean isRunnableStopped()
1980 {
1981     return isRunnableStopped();
1982 }
1983
1984 /**
1985 * It gets the sensor values
1986 */
1987 public JSONObject getSensorValues()
1988 {
1989     return getSensorValues();
1990 }
1991
1992 /**
1993 * It updates the sensor values
1994 */
1995 public void updateSensorValues()
1996 {
1997     updateSensorValues();
1998 }
1999
2000 /**
2001 * It stops the runnable
2002 */
2003 public void stopRunnable()
2004 {
2005     stopRunnable();
2006 }
2007
2008 /**
2009 * It starts the runnable
2010 */
2011 public void startRunnable()
2012 {
2013     startRunnable();
2014 }
2015
2016 /**
2017 * It checks if the runnable is stopped
2018 */
2019 public boolean isRunnableStopped()
2020 {
2021     return isRunnableStopped();
2022 }
2023
2024 /**
2025 * It gets the sensor values
2026 */
2027 public JSONObject getSensorValues()
2028 {
2029     return getSensorValues();
2030 }
2031
2032 /**
2033 * It updates the sensor values
2034
```

```

388     {
389         String values =
390         getURL("http://CENSURED/sensor_values");
391         // return a JSONObject constructed by JSONTokener, which takes
392         // a source string and extracts characters and tokens from it
393         return new JSONObject(new JSONTokener(values));
394     }
395     catch (Exception e)
396     {
397         Log.e(LIVE_CARD_TAG, "Failed to get sensor values", e);
398         return null;
399     }
400 }
401
402 // Get the new data points that we will need to graph
403
404 /**
405 * This function computes the points that have to be plotted
406 *
407 * @param sensor , the name of sensor (pH or Temperature)
408 * @param last_timestamp , the previous value of timestamp
409 * @return JSONArray, return a list of values that corresponds to the
410 * points that have to be plotted
411 */
412 private JSONArray getDataPoints(String sensor, double last_timestamp)
413 {
414     try
415     {
416         String url =
417         "http://CENSURED/graphing_data?sensor=" +
418         sensor + "&last_timestamp=" + String.valueOf(last_timestamp);
419         String values = getURL(url);
420         return new JSONArray(new JSONTokener(values));
421     }
422     catch (Exception e)
423     {
424         Log.e(LIVE_CARD_TAG, "Failed to get data points", e);
425         return null;
426     }
427 }
428
429 /**
430 * This method gets data from the given url website
431 *
432 * @param _url, url in which the data are contained
433 * @return String, contained data from the given url
434 * @throws IOException if an IO exception occurred during the download
435 */
436 private String getURL(String _url) throws IOException
437 {

```

```

        URL url = new URL(_url);
438    InputStream is = url.openStream();
439    int ptr;
440    StringBuffer buffer = new StringBuffer();
441    while ((ptr = is.read()) != -1)
442    {
443        buffer.append((char)ptr);
444    }
445    return buffer.toString();
446}

447 /**
448 * This runnable updates the graphs of pH and Temperature
449 */
450 private class UpdateSensorGraphsRunnable implements Runnable {
451     private boolean mIsStopped = false;
452     public void run()
453     {
454         if (!mIsStopped())
455         {
456             // Loop through each of the sensors
457             for (String curr_sensor : mSensors) {
458                 ArrayList<DataPoint> curr_data =
459                     mSensorGraphData.get(curr_sensor);
460                 double lastTimestamp = curr_data.size() > 0 ?
461                     curr_data.get(curr_data.size() - 1).getTimestamp() : 0.;
462                 // Get a list of the new data points for this sensor
463                 JSONArray newPoints = getDataPoints(curr_sensor, lastTimestamp);
464                 for (int j = 0; j < newPoints.length(); j++) {
465                     // Save each data point
466                     try {
467                         JSONObject point = newPoints.getJSONObject(j);
468                         double timestamp = (Double) point.get("timestamp");
469                         double value = (Double) point.get("value");
470                         curr_data.add(new DataPoint(timestamp, value));
471                     } catch (JSONException e) {
472                         Log.e(LIVE_CARD_TAG, "JSON error", e);
473                     }
474                 }
475                 // Clear out points that are over an hour old
476                 while (true) {
477                     if (curr_data.size() > 0 && curr_data.get(0).getTimestamp()
478                         < (curr_data.get(curr_data.size() - 1)
479                             .getTimestamp() - 3600) {
480                         Log.i(LIVE_CARD_TAG, "Deleting old data point");
481                         curr_data.remove(0);
482                     } else {
483                         break;
484                     }
485                 }
486             }
487             // If there are no points don't show a graph

```

```
486     if (curr_data.size() == 0) {
487         continue;
488     }
489
490     // Store the timestamps and values in separate arrays to graph
491     ArrayList<Double> timestamps = new ArrayList<Double>();
492     ArrayList<Double> values = new ArrayList<Double>();
493
494     for (DataPoint curr_point : curr_data) {
495         timestamps.add(curr_point.getTimestamp());
496         values.add(curr_point.getValue());
497     }
498
499     mSensorAverage.put(curr_sensor, computeAverage(values));
500
501     // Scale the timestamp data
502     double maxTimestamp = Collections.max(timestamps);
503     double minTimestamp = Collections.min(timestamps);
504     maxTimestamp *= 1.2;
505     minTimestamp *= 0.8;
506     double intervalSize = maxTimestamp - minTimestamp;
507     for (int i1 = 0; i1 < timestamps.size(); i1++) {
508         double currTimestamp = timestamps.get(i1);
509         currTimestamp -= minTimestamp;
510         currTimestamp /= intervalSize;
511         currTimestamp *= 100;
512         timestamps.set(i1, currTimestamp);
513     }
514
515     // Scale the value data
516     double maxVal = Collections.max(values);
517     double minVal = Collections.min(values);
518     maxVal *= 1.2;
519     minVal *= 0.8;
520     intervalSize = maxVal - minVal;
521     for (int i1 = 0; i1 < values.size(); i1++) {
522         double currVal = values.get(i1);
523         currVal -= minVal;
524         currVal /= intervalSize;
525         currVal *= 100;
526         values.set(i1, currVal);
527     }
528
529     // Data xData = Data.newData(timestamps);
530     Data yData = Data.newData(values);
531     Plot plot = Plots.newPlot(yData);
532     LineChart lineChart = GCharts.newLineChart(plot);
533     lineChart.setSize(400, 200);
534     lineChart.addYAxisLabels(AxisLabelsFactory
535         .newNumericRangeAxisLabels(minVal, maxVal));
```

```
536         mCurrentSensorGraphs.put(curr_sensor,
537             getBitmapFromURL(lineChart.toURLString()));
538     }
540
541     mCallback.SetSensorAvg(mSensorAverage);
542     mCallback.setSensorGraphs(mCurrentSensorGraphs);
543     Log.i("Main Service", "Graphs updated");
544
545     // restart it after a given amount of time
546     mHandler.postDelayed(mUpdateSensorGraphsRunnable,
547         GRAPH_UPDATE_DELAY_MILLIS);
548 }
549
550
551     private double computeAverage(ArrayList<Double> values) {
552         double sum = 0.0;
553         if(!values.isEmpty()){
554             for(Double value : values){
555                 sum += value;
556             }
557             return sum/values.size();
558         }
559         return sum;
560     }
561
562 /**
563  * @return mIsStopped, true if the runnable is stopped, false otherwise
564  */
565     public boolean isStopped()
566     {
567         return mIsStopped;
568     }
569
570 /**
571  * It is the setter that allows to stop the runnable
572  *
573  * @param isStopped, true if the user wishes to stop the runnable,
574  * false otherwise
575  */
576     public void setStop(boolean isStopped)
577     {
578         this.mIsStopped = isStopped;
579     }
580
581 /**
582  * Runnable that implements the task for downloading beating image */
583     public class ImageDownloader implements Runnable {
584         private String url;
```

```
584     private Context c;
585     private boolean mIsStopped = false;
586
587     /** Class constructor of ImageDownloader runnable
588      *
589      * @param url, url link of image
590      * @param c, is the context in which the request of downloading image
591      *           has been sent
592      */
593
594     public ImageDownloader(String url, Context c)
595     {
596         this.url = url;
597         this.c = c;
598     }
599
600     /** It implements the task of ImageDownloader runnable
601      *
602      * @see
603      */
604
605     @Override
606     public void run()
607     {
608         if (!mIsStopped)
609         {
610             bmp = getBitmapFromURL(url);
611
612             mCallback.setBMP(bmp);
613             Log.i(BEATING_TAG, "bmp settato");
614         }
615     }
616
617     /** It is the getter function that shows the status of ImageDownloader
618      * runnable
619      *
620      * @return mIsStopped, true if the runnable is stopped, false otherwise
621      */
622     public boolean isStopped()
623     {
624         return mIsStopped;
625     }
626
627     /** It is the setter that allows to stop the runnable
628      *
629      * @param isStopped, true if the user wishes to stop the runnable,
630      *           false otherwise
631      */
632     public void setIsStopped(boolean isStopped)
633     {
634         this.mIsStopped = isStopped;
```

```
        }

634    }

635    /**
636     * @param urlLink where the image is stored
637     * @return Bitmap which contains the image of the graph
638     * @throws java.io.IOException if an IO exception occurred during the download
639     */
640

641    public static Bitmap getBitmapFromURL(String urlLink){
642        try{
643            Log.i(BEATING_TAG, "start downloading image ");
644            long startTime = System.currentTimeMillis();
645            URL url = new URL(urlLink);
646            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
647            connection.setDoInput(true);
648            connection.connect();
649            InputStream inputStream = connection.getInputStream();
650            Log.i(BEATING_TAG, "download completed in "
651                  + ((System.currentTimeMillis() - startTime) / 1000)
652                  + " sec");
653            return BitmapFactory.decodeStream(inputStream);
654        } catch (IOException e) {
655            e.printStackTrace();
656            Log.e(BEATING_TAG, e.getMessage());
657            return null;
658        }
659    }

660

661

662    // Runnable that updates the microscope video
663    private class UpdateMicroscopeVideoRunnable implements Runnable {
664        private boolean mIsStopped = false;
665        public void run() {
666            if (!mIsStopped) {
667                getMicroscopeVideo();
668                mHandler.postDelayed(mUpdateMicroscopeVideoRunnable,
669                                      VIDEO_UPDATE_DELAY_MILLIS);
670            }
671        }
672        public boolean isStopped() {
673            return mIsStopped;
674        }
675        public void setStop(boolean isStopped) {
676            this.mIsStopped = isStopped;
677        }
678    }

679

680    // Pull the microscope video from a URL
681    private void getMicroscopeVideo() {
```

```

682     try {
683         URL url = new URL("http://CENSURED/video/view");
684         long startTime = System.currentTimeMillis();
685         Log.i(VIDEO_TAG, "video download beginning: "+url);
686         URLConnection ucon = url.openConnection();
687         ucon.setReadTimeout(0);
688         ucon.setConnectTimeout(0);
689         // Define InputStreams to read from the URLConnection.
690         InputStream is = ucon.getInputStream();
691         BufferedInputStream inStream = new BufferedInputStream(is, 1024*5);
692         File file = new File(TEMP_VIDEO_FILE_NAME);
693
694         FileOutputStream outStream = new FileOutputStream(file);
695
696         FileLock lock = outStream.getChannel().lock();
697         byte[] buff = new byte[1024*5];
698         // Read bytes (and store them) until there is nothing more to read(-1)
699         int len;
700         while ((len = inStream.read(buff)) != -1) {
701             outStream.write(buff,0,len);
702         }
703         // Clean up
704
705         outStream.flush();
706         lock.release();
707         outStream.close();
708         inStream.close();
709         Log.i(VIDEO_TAG, "download completed in "
710               + ((System.currentTimeMillis() - startTime) / 1000)
711               + " sec");
712     }
713     catch (IOException e) {
714         Log.e(VIDEO_TAG, "Failed to download microscope video", e);
715     }
716 }
717
718 private class UpdateElectrovalvesStatus implements Runnable {
719     private boolean mIsStopped = false;
720
721     public void run(){
722         if (!mIsStopped){
723             try {
724                 getElectrovalvesStatus();
725             } catch (IOException e) {
726                 e.printStackTrace();
727             } catch (InterruptedException e) {
728                 e.printStackTrace();
729             }
730             Log.i(LIVE_CARD_TAG, "Status Updated");
731         }
732     }
733 }
```

```
        }

732    }

733    public boolean isStopped(){
734        return mIsStopped;
735    }

736    public void setStop(boolean isStopped){
737        this.mIsStopped = isStopped;
738    }

739}

740

741    public static void getElectrovalvesStatus() throws IOException,
742                                                    InterruptedException {
743        for(int index = 0; index < Electrovalves.ELECTROVALVES_NUMBER; index++){
744            String res = HTTPget(Electrovalves.URL_ELECTROVALVES_BASE + "EV" +
745                                  Integer.toString(index+1));
746            if(res.equals(" True")){
747                Electrovalves.setElectrovalvesStatus(index,true);
748            }
749            else if(res.equals(" False")){
750                Electrovalves.setElectrovalvesStatus(index,false);
751            }
752            Log.i("EV"+Integer.toString(index+1), res);
753        }
754    }

755}

756

757

758    private static String HTTPget(String link) throws IOException,
759                                                    InterruptedException {
760        URL url = new URL(link);
761        HttpURLConnection mURLConnection = (HttpURLConnection) url.openConnection();
762        InputStream in = new BufferedInputStream(mURLConnection.getInputStream());
763        int ch;
764        StringBuffer b = new StringBuffer();
765        while ((ch = in.read()) != -1){
766            b.append((char) ch);
767        }
768

769        String mResult = new String(b);
770        return mResult;
771    }

772}

773

774    // convert inputstream to String
775    private static String convertInputStreamToString(InputStream inputStream)
776                                                    throws IOException{
777        BufferedReader bufferedReader = new BufferedReader(
778                                new InputStreamReader(inputStream));
779        String line = "";
780    }
```

```

780     String result = "";
781     while((line = bufferedReader.readLine()) != null)
782         result += line;
783
784     inputStream.close();
785     return result;
786 }
787
788
789 }
```

Listing A.18: AppDrawer.java

```

// This drawer has been designed using the chronometer example provided by Google
2 package com.google.android.glass.sample.klabinterface;

4 import com.google.android.glass.timeline.DirectRenderingCallback;
5 import com.jjoe64.graphview.GraphView;
6
7 import android.content.Context;
8 import android.content.Intent;
9 import android.graphics.Bitmap;
10 import android.graphics.Canvas;
11 import android.os.Environment;
12 import android.os.Handler;
13 import android.os.SystemClock;
14 import android.util.Log;
15 import android.view.SurfaceHolder;
16 import android.view.View;

18 import java.util.Map;

20 /**
21 * {@link DirectRenderingCallback} used to draw the KlabInterface on the timeline
22 * {@link com.google.android.glass.timeline.LiveCard}.
23 * Rendering requires that:
24 * <ol>
25 * <li>a {@link SurfaceHolder} has been created through monitoring the
26 *      {@link SurfaceHolder.Callback#(SurfaceHolder)} and
27 *      {@link SurfaceHolder.Callback#(SurfaceHolder)} callbacks.
28 * <li>rendering has not been paused (defaults to rendering) through monitoring the
29 *      {@link com.google.android.glass.timeline.DirectRenderingCallback#}
30 *      renderingPaused(SurfaceHolder, boolean)} callback.
31 * </ol>
32 * As this class uses an inflated {@link View} to draw on the
33 * {@link SurfaceHolder}'s
34 * {@link Canvas}, monitoring the
35 * {@link SurfaceHolder.Callback#(SurfaceHolder, int, int, int)}} callback is also
```

```
36     * required to properly measure and layout the {@link View}'s dimension.  
37     */  
38     public class AppDrawer implements DirectRenderingCallback {  
39         /** INT associated to the Menu view request */  
40         private static final int MENU = 0;  
41         /** INT associated to the PH view request */  
42         private static final int PH = 1;  
43         /** INT associated to the Menu view request */  
44         private static final int TEMPERATURE = 2;  
45         /** INT associated to the Video view request */  
46         private static final int VIDEO = 3;  
47         /** INT associated to the Beating view request */  
48         private static final int BEATING = 4;  
49  
50         /** Bitmap in which the beating image is stored */  
51         private Bitmap bmp;  
52         private boolean mReady;  
53  
54         private static final String TAG = AppDrawer.class.getSimpleName();  
55         /** View object of the GlassWear Main window */  
56         private final MainView mMainView;  
57         /** View object of the GlassWear Beating window */  
58         private final BeatingView mBeatingView;  
59         /** View object of the GlassWear pH window */  
60         private final PHViewer mPhViewer;  
61         /** View object of the GlassWear Temperature window */  
62         private final TemperatureView mTemperatureView;  
63  
64         private SurfaceHolder mHolder;  
65         private boolean mRenderingPaused;  
66  
67         /** Hash table in which the sensors graphs are stored */  
68         private Map<String, Bitmap> mCurrentSensorGraphs;  
69         /** Hash table in which the sensors values are stored */  
70         private Map<String, Double> mCurrentSensorValues;  
71         /** Hash table in which the sensors average values are stored */  
72         private Map<String, Double> mSensorAverage;  
73         private GraphView mGraphView;  
74  
75         private final MainView.Listener mMainListener = new MainView.Listener() {  
76  
77             @Override  
78             public void onChange() {  
79                 updateRenderingState();  
80             }  
81         };  
82  
83         /** Defines the Listener of pH viewer, it is used to communicate with
```

```
* that viewer and allowing its viewing */
86 private final PHViewer.Listener mPhListener = new PHViewer.Listener(){
87     @Override
88     public void onChange(){
89         updateRenderingState();
90     }
91 };
92
93 /**
94  * Defines the Listener of Beating viewer, it is used to communicate with
95  * that viewer and allowing its viewing */
96 private final BeatingView.Listener mBeatingListener =
97     new BeatingView.Listener() {
98
99     @Override
100    public void onChange() {
101        updateRenderingState();
102    }
103 };
104
105 /**
106  * Defines the Listener of Temperature viewer, it is used to communicate with
107  * that viewer and allowing its viewing */
108 private final TemperatureView.Listener mTemperatureListener =
109     new TemperatureView.Listener() {
110
111     @Override
112     public void onChange() {
113         updateRenderingState();
114     }
115 };
116
117 /**
118  * Defines the ManageBitmap of Beating viewer, it is used to communicate with
119  * that viewer in order to update the bitmap to be displayed*/
120 private final BeatingView.ManageBitmap mBeatingBitmap =
121     new BeatingView.ManageBitmap(){
122
123     @Override
124     public Bitmap getBitmap() {
125         return bmp;
126     }
127
128     public boolean isReady(){
129         return mReady;
130     }
131 };
132
133 /**
134  * Defines the ManageDataGraph of PH viewer, it is used to communicate with
135  * that viewer in order to update the hash map which contains the graph to
136  * be displayed*/
137 private final PHViewer.ManageBitmap mPHManageDataGraph =
138     new PHViewer.ManageBitmap(){
```

```
134     public Bitmap getBitmap() {
135         return mCurrentSensorGraphs.get("pH");
136     }
138
139     public double getAvg(){return mSensorAverage.get("pH");}
140
141     public boolean isReady(){
142         return mReady;
143     }
144 }
145
146 /**
147 * Defines the ManageDataGraph of PH viewer, it is used to communicate with
148 * that viewer in order to update the hash map which contains the graph
149 * to be displayed*/
150 private final TemperatureView.ManageBitmap mTemperatureManageDataGraph =
151             new TemperatureView.ManageBitmap(){
152
153     public Bitmap getBitmap() {
154         return mCurrentSensorGraphs.get("pH");
155     }
156
157     public double getAvg(){return mSensorAverage.get("Temperature");}
158
159     public boolean isReady(){
160         return mReady;
161     }
162 }
163
164
165     public AppDrawer.(Context context) {
166         this(new MainView(context), new BeatingView(context),
167               new PHViewer(context), new TemperatureView(context));
168     }
169
170
171     public AppDrawer.(MainView countDownView, BeatingView chronometerView,
172                     PHViewer phViewer, TemperatureView temperatureView) {
173         mMainView = countDownView;
174         mMainView.setListener(mMainListener);
175
176         mBeatingView = chronometerView;
177         mBeatingView.setListener(mBeatingListener);
178         mBeatingView.setManageBPM(mBeatingBitmap);
179
180         mPhViewer = phViewer;
181         mPhViewer.setListener(mPhListener);
182         mPhViewer.setManageBPM(mPHManageDataGraph);
183
184         mTemperatureView = temperatureView;
```

```

    mTemperatureView.setListener(mTemperatureListener);
184    mTemperatureView.setManageBPM(mTemperatureManageDataGraph);

186    mReady = false;
}

188 /**
190 * Uses the provided the width and height to measure and layout the
191 * inflated View.
192 */
193 @Override
194 public void surfaceChanged(SurfaceHolder holder, int format, int width,
195                           int height) {
196     // Measure and layout the view with the canvas dimensions.
197     int measuredWidth = View.MeasureSpec.makeMeasureSpec(width,
198                 View.MeasureSpec.EXACTLY);
199     int measuredHeight = View.MeasureSpec.makeMeasureSpec(height,
200                 View.MeasureSpec.EXACTLY);

201     mMainView.measure(measuredWidth, measuredHeight);
202     mMainView.layout(0, 0, mMainView.getMeasuredWidth(),
203                     mMainView.getMeasuredHeight());

204     mBeatingView.measure(measuredWidth, measuredHeight);
205     mBeatingView.layout(0, 0, mBeatingView.getMeasuredWidth(),
206                         mBeatingView.getMeasuredHeight());

207     mPhViewer.measure(measuredWidth, measuredHeight);
208     mPhViewer.layout(0, 0, mPhViewer.getMeasuredWidth(),
209                      mPhViewer.getMeasuredHeight());

210     mTemperatureView.measure(measuredWidth, measuredHeight);
211     mTemperatureView.layout(0, 0, mTemperatureView.getMeasuredWidth(),
212                           mTemperatureView.getMeasuredHeight());
213 }

214 /**
215 * Keeps the created SurfaceHolder and updates this class' rendering state.
216 */
217 @Override
218 public void surfaceCreated(SurfaceHolder holder) {
219     // The creation of a new Surface implicitly resumes the rendering.
220     mRenderingPaused = false;
221     mHolder = holder;
222     updateRenderingState();
223 }

224 /**
225 * Removes the SurfaceHolder used for drawing and stops rendering.

```

```
232     */
233     @Override
234     public void surfaceDestroyed(SurfaceHolder holder) {
235         mHolder = null;
236         updateRenderingState();
237     }
238
239     /**
240      * Updates this class rendering state according to the provided paused flag.
241      */
242     @Override
243     public void renderingPaused(SurfaceHolder holder, boolean paused) {
244         mRenderingPaused = paused;
245         updateRenderingState();
246     }
247
248     /**
249      * Starts or stops rendering according to the LiveCard's state.
250      */
251     private void updateRenderingState() {
252         if (mHolder != null && !mRenderingPaused) {
253             switch (AppManager.getInstance().getState())
254             {
255                 case MENU:
256                     draw(mMainView);
257                     mBeatingView.stop();
258                     mPhViewer.stop();
259                     mTemperatureView.stop();
260                     mMainView.start();
261                     break;
262                 case BEATING:
263                     draw(mBeatingView);
264                     mMainView.stop();
265                     mPhViewer.stop();
266                     mTemperatureView.stop();
267                     mBeatingView.start();
268                     break;
269                 case PH:
270                     draw(mPhViewer);
271                     mMainView.stop();
272                     mBeatingView.stop();
273                     mTemperatureView.stop();
274                     mPhViewer.start();
275                     break;
276                 case TEMPERATURE:
277                     draw(mTemperatureView);
278                     mMainView.stop();
279                     mBeatingView.stop();
280                     mPhViewer.stop();
281             }
282         }
283     }
```

```
282         mTemperatureView.start();
283         break;
284     case VIDEO:
285         mTemperatureView.stop();
286         mBeatingView.stop();
287         mMMainView.stop();
288         mPhViewer.stop();
289         break;
290     default:
291         draw(mMainView);
292         mPhViewer.stop();
293         mBeatingView.stop();
294         mTemperatureView.stop();
295         mMMainView.start();
296         break;
297     }
298 }
299
300 } else {
301     mMMainView.stop();
302     mBeatingView.stop();
303     mPhViewer.stop();
304     mTemperatureView.stop();
305 }
306
307 /**
308 * Draws the view in the SurfaceHolder's canvas.
309 */
310 private void draw(View view) {
311
312     Canvas canvas;
313
314     try {
315         canvas = mHolder.lockCanvas();
316     } catch (Exception e) {
317         Log.e(TAG, "Unable to lock canvas: " + e);
318         return;
319     }
320     if (canvas != null) {
321
322         view.draw(canvas);
323         mHolder.unlockCanvasAndPost(canvas);
324     }
325 }
326
327 /**
328 * Setter for the beating graph
329 *
330 * @param bmp , Bitmap which contains the beating graph
331 */
```

```
330     */
331     public void setBMP(Bitmap bmp){
332
333         this.bmp = bmp;
334         this.mReady = true;
335         Log.i("DRAWER", "bmp settato");
336     }
337
338     /**
339      * Setter for the sensors Graphs hash map
340      *
341      * @param sensorGraphs , Map<String,Double> sensorGraphs
342      * which contains the hash
343      * map with graphs of the sensors
344      */
345     public void setSensorGraphs( Map<String,Bitmap> sensorGraphs ){
346         this.mCurrentSensorGraphs = sensorGraphs;
347     }
348
349     /**
350      * Setter for the sensors viewer hash map
351      *
352      * @param graphView , Map<String,Double> sensorGraphs
353      * which contains the hash
354      * map with viewer of the sensors
355      */
356     public void setPHGraphViewer( GraphView graphView ){
357         this.mGraphView = graphView;
358     }
359
360     /**
361      * Setter for the sensors value hash map
362      *
363      * @param currentSensorValues , Map<String,Double> currentSensorValues
364      * which contains the hash
365      * map with the current value of the sensors
366      */
367     public void setSensorValues( Map<String,Double> currentSensorValues ){
368         this.mCurrentSensorValues = currentSensorValues;
369     }
370
371     /**
372      * Setter for the sensors average values hash map
373      *
374      * @param sensorAvg , Map<String,Double> currentSensorValues
375      * which contains the hash
376      * map with the average values of the sensors
377      */
378     public void SetSensorAvg (Map<String,Double> sensorAvg){
379         this.mSensorAverage = sensorAvg;
380     }
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266

```

380 }

Listing A.19: MainView.java

```
package com.google.android.glass.sample.klabinterface;

import android.content.Context;
import android.media.AudioManager;
import android.media.SoundPool;
import android.os.Handler;
import android.os.SystemClock;
import android.util.AttributeSet;
import android.view.LayoutInflater;
import android.widget.FrameLayout;
import android.widget.TextView;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;

public class MainView extends FrameLayout {

    /**
     * Interface to listen for changes in the countdown.
     */
    public interface Listener {

        /**
         * Notified when the countdown is finished.
         */
        public void onChange();
    }

    /** Time delimiter specifying when the second component is fully shown. */
    private static final long DELAY_MILLIS = 40;

    private final TextView timeText;

    private final Handler mHandler = new Handler();
    private final Runnable mUpdateViewRunnable = new Runnable() {

        @Override
        public void run() {
            if (mRunning) {
                updateView();
                postDelayed(mUpdateViewRunnable, DELAY_MILLIS);
            }
        }
    }
}
```

```
        };

46
    private Listener mListener;
48
    private boolean mRunning = false;

50
    public MainView(Context context) {
        this(context, null, 0);
52    }

54
    public MainView(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
56    }

58
    public MainView(Context context, AttributeSet attrs, int style) {
        super(context, attrs, style);
60        LayoutInflater.from(context).inflate(R.layout.live_card_layout, this);
        timeText = (TextView) findViewById(R.id.timestamp);
62    }

64
    /**
     * Sets a {@link Listener}.
     */
66
    public void setListener(Listener listener) {
68        mListener = listener;
69    }

70
    /**
     * Returns the set {@link Listener}.
     */
72
    public Listener getListener() {
74        return mListener;
75    }

78
    @Override
    public boolean postDelayed(Runnable action, long delayMillis) {
80        return mHandler.postDelayed(action, delayMillis);
81    }

82
    /**
     * Starts the countdown animation if not yet started.
     */
84
    public void start() {
86        if (!mRunning) {
87            postDelayed(mUpdateViewRunnable, 0);
88        }
89        mRunning = true;
90    }

92
    /**

```

```

94     * Stops the chronometer.
95     */
96     public void stop() {
97         if (mRunning) {
98             removeCallbacks(mUpdateViewRunnable);
99             // mStarted = false;
100        }
101        mRunning = false;
102    }
103
104    void updateView() {
105        if (mRunning) {
106            timeText.setText( new SimpleDateFormat("hh:mm a")  

107                .format( new Date()) );
108            if (mListener != null) {
109                mListener.onChange();
110            }
111        }
112    }
113}
114

```

Listing A.20: BeatingView.java

```

package com.google.android.glass.sample.klabinterface;  

2   import android.content.Context;  

4   import android.graphics.Bitmap;  

5   import android.graphics.drawable.Drawable;  

6   import android.os.Handler;  

7   import android.text.Layout;  

8   import android.util.AttributeSet;  

9   import android.util.Log;  

10  import android.view.LayoutInflater;  

11  import android.widget.FrameLayout;  

12  import android.widget.ImageView;  

13  import android.widget.TextView;  

14
15
16  public class BeatingView extends FrameLayout {
17
18      boolean mState = false;
19      private Bitmap bmp;
20
21      /**
22       * Interface to listen for changes on the view layout.
23       */
24      public interface Listener {
25          /** Notified of a change in the view. */
26          public void onChange();
27      }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114

```

```
26     }
27
28     /** About 24 FPS, visible for testing. */
29     static final long DELAY_MILLIS = 41;
30
31
32     private ImageView beatingImage;
33     private final TextView mTitle;
34
35
36     private final Handler mHandler = new Handler();
37     private final Runnable mUpdateTextRunnable = new Runnable() {
38
39         @Override
40         public void run() {
41             if (mRunning) {
42                 updateText();
43                 postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
44             }
45         }
46
47     };
48
49     private boolean mRunning;
50
51
52     public interface ManageBitmap{
53         public Bitmap getBitmap();
54
55         public boolean isReady();
56     }
57
58     private Listener mChangeListener;
59
60     private ManageBitmap mManage;
61
62     public BeatingView(Context context) {
63         this(context, null, 0);
64     }
65
66     public BeatingView(Context context, AttributeSet attrs) {
67         this(context, attrs, 0);
68     }
69
70     public BeatingView(Context context, AttributeSet attrs, int style) {
71         super(context, attrs, style);
72         LayoutInflator.from(context).inflate(R.layout.buso_layout, this);
73         mTitle = (TextView) findViewById(R.id.message);
74         beatingImage = (ImageView) findViewById(R.id.image_left);
75         mTitle.setText("Beating");
76         int id = getResources().getIdentifier(
77             "com.google.android.glass.sample.stopwatch:drawable/layout_color",
78             null, null);
```

```
    beatingImage.setImageResource(id);
76 }

78 /**
 * Sets a Listener
 */
80
81     public void setListener(Listener listener) {
82         mChangeListener = listener;
83     }
84
85 /**
86 * Returns the set Listener.
87 */
88     public Listener getListener() {
89         return mChangeListener;
90     }
91
92 /**
93 * Starts the BeatingView.
94 */
95     public void start() {
96         if (!mRunning) {
97             postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
98         }
99         mRunning = true;
100    }
101
102 /**
103 * Stops the BeatingView.
104 */
105     public void stop() {
106         if (mRunning) {
107             removeCallbacks(mUpdateTextRunnable);
108         }
109         mRunning = false;
110    }
111
112 @Override
113     public boolean postDelayed(Runnable action, long delayMillis) {
114         return mHandler.postDelayed(action, delayMillis);
115     }
116
117 @Override
118     public boolean removeCallbacks(Runnable action) {
119         mHandler.removeCallbacks(action);
120         return true;
121     }
122 /**
123 * Sets a {@link Listener}.
124 */
```

```
124 */  
125  
126     public void setManageBPM(ManageBitmap manager) {  
127         mManage = manager;  
128     }  
129  
130     /**  
131      * Updates the value of the chronometer, visible for testing.  
132      */  
133  
134     void updateText() {  
135         if (mManage.isReady())  
136         {  
137             Log.i("viewer", "ce prova");  
138             bmp = mManage.getBitmap();  
139             beatingImage.setImageBitmap(bmp);  
140         }  
141         if (mChangeListener != null) {  
142             mChangeListener.onChange();  
143         }  
144     }  
145 }
```

Listing A.21: PHViewer.java

```
26     * Interface to listen for changes on the view layout.  
27     */  
28  
29     public interface Listener {  
30         /** Notified of a change in the view. */  
31         public void onChange();  
32     }  
33  
34     static final long DELAY_MILLIS = 41;  
35  
36     private ImageView beatingImage;  
37     private final TextView mTextTitle;  
38     private TextView AvgView;  
39  
40     private final Handler mHandler = new Handler();  
41     private final Runnable mUpdateTextRunnable = new Runnable() {  
42  
43         @Override  
44         public void run() {  
45             if (mRunning) {  
46                 updateText();  
47                 postDelayed(mUpdateTextRunnable, DELAY_MILLIS);  
48             }  
49         }  
50     };  
51  
52     private boolean mRunning;  
53  
54     public interface ManageBitmap{  
55         public Bitmap getBitmap();  
56  
57         public boolean isReady();  
58  
59         public double getAvg();  
60     }  
61  
62     private Listener mChangeListener;  
63  
64     private ManageBitmap mManage;  
65  
66     public PHViewer(Context context) {  
67         this(context, null, 0);  
68     }  
69  
70     public PHViewer(Context context, AttributeSet attrs) {  
71         this(context, attrs, 0);  
72     }  
73  
74     public PHViewer(Context context, AttributeSet attrs, int style) {  
75         super(context, attrs, style);  
76         LayoutInflater.from(context).inflate(R.layout.buso_layout, this);  
77     }
```

```
    beatingImage = (ImageView) findViewById(R.id.image_left);
76   mTextTitle = (TextView) findViewById(R.id.message);
77   AvgView = (TextView) findViewById(R.id.avg);
78   mTextTitle.setText("PH");
79   int id = getResources().getIdentifier(
80       "com.google.android.glass.sample.stopwatch:drawable/layout_color"
81       ,null,null);
82   beatingImage.setImageResource(id);
83 }
84
85 /**
86  * Sets a Listener.
87  */
88 public void setListener(Listener listener) {
89     mChangeListener = listener;
90 }
91
92 /**
93  * Returns the set Listener.
94  */
95 public Listener getListener() {
96     return mChangeListener;
97 }
98
99 /**
100 * Starts the chronometer.
101 */
102 public void start() {
103     if (!mRunning) {
104         postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
105     }
106     mRunning = true;
107 }
108
109 /**
110 * Stops the chronometer.
111 */
112 public void stop() {
113     if (mRunning) {
114         removeCallbacks(mUpdateTextRunnable);
115     }
116     mRunning = false;
117 }
118
119 @Override
120 public boolean postDelayed(Runnable action, long delayMillis) {
121     return mHandler.postDelayed(action, delayMillis);
122 }
```

```

124     @Override
125     public boolean removeCallbacks(Runnable action) {
126         mHandler.removeCallbacks(action);
127         return true;
128     }
129
130     /**
131      * Sets a Listener.
132      */
133     public void setManageBPM(ManageBitmap manager) {
134         mManage = manager;
135     }
136
137
138     /**
139      * Updates the value of the chronometer, visible for testing.
140      */
141     void updateText() {
142         if(mManage.isReady())
143         {
144             bmp = mManage.getBitmap();
145             AvgView.setText(Double.toString(mManage.getAvg()));
146             beatingImage.setImageBitmap(bmp);
147         }
148         if (mChangeListener != null) {
149             mChangeListener.onChange();
150         }
151     }
152 }
```

Listing A.22: TemperatureView.java

```

1 package com.google.android.glass.sample.klabinterface;
2
3     import android.content.Context;
4     import android.graphics.Bitmap;
5     import android.graphics.drawable.Drawable;
6     import android.os.Handler;
7     import android.text.Layout;
8     import android.util.AttributeSet;
9     import android.util.Log;
10    import android.view.LayoutInflater;
11    import android.widget.FrameLayout;
12    import android.widget.ImageView;
13    import android.widget.TextView;
14
15
16
17    public class TemperatureView extends FrameLayout {
18        private Bitmap bmp;
```

```
19     private TextView AvgView;  
21  
22     /**  
23      * Interface to listen for changes on the view layout.  
24      */  
25     public interface Listener {  
26         /** Notified of a change in the view. */  
27         public void onChange();  
28     }  
29  
30     /** About 24 FPS, visible for testing. */  
31     static final long DELAY_MILLIS = 41;  
32  
33     private ImageView beatingImage;  
34     private final TextView mTitle;  
35  
36     private final Handler mHandler = new Handler();  
37     private final Runnable mUpdateTextRunnable = new Runnable() {  
38  
39         @Override  
40         public void run() {  
41             if (mRunning) {  
42                 updateText();  
43                 postDelayed(mUpdateTextRunnable, DELAY_MILLIS);  
44             }  
45         }  
46     };  
47  
48     private boolean mRunning;  
49  
50     public interface ManageBitmap{  
51         public Bitmap getBitmap();  
52  
53         public boolean isReady();  
54  
55         public double getAvg();  
56     }  
57     private Listener mChangeListener;  
58  
59     private ManageBitmap mManage;  
60  
61     public TemperatureView(Context context) {  
62         this(context, null, 0);  
63     }  
64  
65     public TemperatureView(Context context, AttributeSet attrs) {  
66         this(context, attrs, 0);  
67     }
```

```
69     public TemperatureView(Context context, AttributeSet attrs, int style) {
70         super(context, attrs, style);
71         LayoutInflator.from(context).inflate(R.layout.buso_layout, this);
72         beatingImage = (ImageView) findViewById(R.id.image_left);
73         mTitle = (TextView) findViewById(R.id.message);
74         AvgView = (TextView) findViewById(R.id.avg);
75
76         mTitle.setText("Temperature");
77         int id = getResources().getIdentifier(
78             "com.google.android.glass.sample.stopwatch:drawable/layout_color"
79             , null, null);
80         beatingImage.setImageResource(id);
81     }
82
83     /**
84      * Sets a Listener.
85      */
86     public void setListener(Listener listener) {
87         mChangeListener = listener;
88     }
89
90     /**
91      * Returns the set Listener.
92      */
93     public Listener getListener() {
94         return mChangeListener;
95     }
96
97     /**
98      * Starts the chronometer.
99      */
100    public void start() {
101        if (!mRunning) {
102            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
103        }
104        mRunning = true;
105    }
106
107    /**
108     * Stops the chronometer.
109     */
110    public void stop() {
111        if (mRunning) {
112            removeCallbacks(mUpdateTextRunnable);
113        }
114        mRunning = false;
115    }
116
117    @Override
118    public boolean postDelayed(Runnable action, long delayMillis) {
```

```

117         return mHandler.postDelayed(action, delayMillis);
    }

119
120     @Override
121     public boolean removeCallbacks(Runnable action) {
122         mHandler.removeCallbacks(action);
123         return true;
    }

125     /**
126      * Sets a Listener.
127      */
128
129     public void setManageBPM(ManageBitmap manager) {
130         mManage = manager;
    }

131
132     void updateText() {
133         if (mManage.isReady())
    {
134             bmp = mManage.getBitmap();
135             AvgView.setText(Double.toString(mManage.getAvg()));
136             beatingImage.setImageBitmap(bmp);
    }

139
140         if (mChangeListener != null) {
141             mChangeListener.onChange();
    }
142     }
}

```

Listing A.23: VideoPlayerActivity.java

```

package com.google.android.glass.sample.klabinterface;

2
import android.app.Activity;
4 import android.content.Intent;
import android.content.res.AssetManager;
6 import android.net.Uri;
import android.os.Bundle;
8 import android.os.Environment;
import android.util.Log;
10 import android.view.Menu;
import android.view.MenuItem;
12 import android.widget.MediaController;
import android.widget.VideoView;

14
import java.io.File;
16 import java.io.FileOutputStream;
import java.io.IOException;
18 import java.io.InputStream;

```

```

import java.io.OutputStream;
20 public class VideoPlayerActivity extends Activity {
    private static final int VIDEO_PLAY_REQUEST_CODE = 200;
22    private static final String TAG = "VIDEO_TAG";
    public void onCreate(Bundle savedInstanceState) {
24        super.onCreate(savedInstanceState);
        String filepath;
26        Bundle extras = getIntent().getExtras();
        if (extras != null)
            filepath = extras.getString("filepath");
        else {
            filepath = copyAsset(VIDEO_FILE_NAME);
        }
28
32        Intent i = new Intent();
34        i.setAction("com.google.glass.action.VIDEOPLAYER");
36        i.putExtra("video_url", filepath);
        startActivityForResult(i, VIDEO_PLAY_REQUEST_CODE);
    }
38    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
40        if (requestCode == VIDEO_PLAY_REQUEST_CODE)
            finish();
    }
42    String copyAsset(String filename) {
44        final String PATH = Environment.getExternalStorageDirectory().toString()
46            + "/myvideoapps/";
        File dir = new File(PATH);
48        if (!dir.exists()) {
            if (!dir.mkdirs()) {
                Log.v(TAG, "ERROR: Creation of directory " + PATH
                    + " on sdcard failed");
                return null;
            } else {
                Log.v(TAG, "Created directory " + PATH + " on sdcard");
            }
        }
50
54        if (!(new File( PATH + filename).exists())) {
56            try {
58                AssetManager assetManager = getAssets();
59                InputStream in = assetManager.open(filename);
60                OutputStream out = new FileOutputStream(PATH + filename);
61                byte[] buf = new byte[1024];
62                int len;
63                while ((len = in.read(buf)) > 0) {
64                    out.write(buf, 0, len);
65                }
66                in.close();
                out.close();
            } catch (IOException e) {

```

```
68         Log.e(TAG, "Was unable to copy " + filename + e.toString());
69         return null;
70     }
71 }
72 return PATH + filename;
73 }
74 }
```

Listing A.24: MenuActivity.java

```
package com.google.android.glass.sample.klabinterface;
2
3 import android.app.Activity;
4 import android.content.Intent;
5 import android.os.Bundle;
6 import android.os.Handler;
7 import android.view.Menu;
8 import android.view.MenuInflater;
9 import android.view.MenuItem;
10 import android.view.View;
11 import android.widget.Toast;
12
13 import java.lang.Runnable;
14
15 /**
16 * Activity showing the stopwatch options menu.
17 */
18 public class MenuActivity extends Activity {
19
20     /** INT associated to the Menu view request */
21     private static final int MENU = 0;
22     /** INT associated to the PH view request */
23     private static final int PH = 1;
24     /** INT associated to the Menu view request */
25     private static final int TEMPERATURE = 2;
26     /** INT associated to the Video view request */
27     private static final int VIDEO = 3;
28     /** INT associated to the Beating view request */
29     private static final int BEATING = 4;
30     /** INT associated to the Electrovalves view request */
31     private static final int ELECTROVALVES = 5;
32
33     private final Handler mHandler = new Handler();
34     private int state = 0;
35
36     @Override
37     public void onAttachedToWindow() {
38         super.onAttachedToWindow();
39         openOptionsMenu();
40     }
41 }
```

```

40    }

42    @Override
43    public boolean onCreateOptionsMenu(Menu menu) {
44        MenuInflater inflater = getMenuInflater();
45        inflater.inflate(R.menu.stopwatch, menu);
46        return true;
47    }

48    @Override
49    public boolean onPreparePanel(int featureId, View view, Menu menu) {
50        AppManager appManager = AppManager.getInstance();

51        boolean initialView = appManager.getState() == MENU;
52        boolean imageView = appManager.getState() == PH ||
53            appManager.getState() == TEMPERATURE ||
54            appManager.getState() == BEATING ||
55            appManager.getState() == ELECTROVALVES ||
56            appManager.getState() == VIDEO;

57        setOptionsMenuState(menu, R.id.action_back, imageView);
58        setOptionsMenuState(menu, R.id.action_view_ph, initialView);
59        setOptionsMenuState(menu, R.id.action_view_temperature, initialView);
60        setOptionsMenuState(menu, R.id.action_view_video, initialView);
61        setOptionsMenuState(menu, R.id.action_view_beating, initialView);
62        setOptionsMenuState(menu, R.id.action_drive_electrovalves, initialView);
63        setOptionsMenuState(menu, R.id.action_toggle_EV1, electrovalveView);
64        setOptionsMenuState(menu, R.id.action_toggle_EV2, electrovalveView);
65        setOptionsMenuState(menu, R.id.action_toggle_EV3, electrovalveView);
66        setOptionsMenuState(menu, R.id.action_toggle_EV4, electrovalveView);
67        setOptionsMenuState(menu, R.id.action_toggle_EV5, electrovalveView);
68        setOptionsMenuState(menu, R.id.action_toggle_EV6, electrovalveView);
69        setOptionsMenuState(menu, R.id.action_toggle_EV7, electrovalveView);
70        setOptionsMenuState(menu, R.id.action_toggle_EV8, electrovalveView);
71        setOptionsMenuState(menu, R.id.action_stop, true);

72        return true;
73    }

74    @Override
75    public boolean onOptionsItemSelected(MenuItem item) {
76        // Handle item selection.
77        switch (item.getItemId()) {
78            case R.id.action_stop:
79                // Stop the service at the end of the message queue for proper
80                // options menu animation. This is only needed when starting
81                // a new Activity or stopping a Service that published a LiveCard.
82                post(new Runnable() {
83                    @Override
84
85
86
87
88

```

```
90         public void run() {
91             stopService(new Intent(MenuActivity.this,
92                             StopwatchService.class));
93         });
94     }
95
96     return true;
97
98     case R.id.action_view_beating:
99         handleViewBeating();
100    return true;
101
102    case R.id.action_back:
103        handleViewBack();
104    return true;
105
106    case R.id.action_view_temperature:
107        handleViewTemperature();
108    return true;
109
110    case R.id.action_view_ph:
111        handleViewPh();
112    return true;
113
114    case R.id.action_view_video:
115        handleViewVideo();
116    return true;
117
118    case R.id.action_drive_electrovalves:
119        handleViewElectrovalves();
120    return true;
121
122    case R.id.action_toggle_EV1:
123        handleToggleEV1();
124    return true;
125
126    case R.id.action_toggle_EV2:
127        handleToggleEV2();
128    return true;
129
130    case R.id.action_toggle_EV3:
131        handleToggleEV3();
132    return true;
133
134    case R.id.action_toggle_EV4:
135        handleToggleEV4();
136    return true;
137
138    case R.id.action_toggle_EV5:
139        handleToggleEV5();
140    return true;
141
142    case R.id.action_toggle_EV6:
143        handleToggleEV6();
144    return true;
145
146    case R.id.action_toggle_EV7:
147        handleToggleEV7();
148    return true;
149
150    case R.id.action_toggle_EV8:
151        handleToggleEV8();
152    return true;
153
154    default:
```

```
138         return super.onOptionsItemSelected(item);
139     }
140 }
141
142 private void handleViewVideo() {
143     Toast.makeText(this, "Video", Toast.LENGTH_SHORT).show();
144     // launch a new thread for starting a new live card
145     mHandler.post(new Runnable() {
146         @Override
147         public void run() {
148             Intent intent = new Intent(MenuActivity.this,
149                 StopwatchService.class);
150             intent.putExtra(Intent.EXTRA_TEXT, "Video");
151             startService(intent);
152         }
153     });
154 }
155
156 private void handleViewTemperature() {
157     Toast.makeText(this, "Temperature", Toast.LENGTH_SHORT).show();
158     // launch a new thread for starting a new live card
159     mHandler.post(new Runnable() {
160         @Override
161         public void run() {
162             Intent intent = new Intent(MenuActivity.this,
163                 StopwatchService.class);
164             intent.putExtra(Intent.EXTRA_TEXT, "Temperature");
165             startService(intent);
166         }
167     });
168 }
169
170 private void handleViewPh() {
171     Toast.makeText(this, "Ph", Toast.LENGTH_SHORT).show();
172     // launch a new thread for starting a new live card
173     mHandler.post(new Runnable() {
174         @Override
175         public void run() {
176             Intent intent = new Intent(MenuActivity.this,
177                 StopwatchService.class);
178             intent.putExtra(Intent.EXTRA_TEXT, "pH");
179             startService(intent);
180         }
181     });
182 }
183
184 private void handleViewBack() {
185     Toast.makeText(this, "Menu", Toast.LENGTH_SHORT).show();
186     // launch a new thread for starting a new live card
```

```
188     mHandler.post(new Runnable() {
189         @Override
190         public void run() {
191             Intent intent = new Intent(MenuActivity.this,
192                 StopwatchService.class);
193             intent.putExtra(Intent.EXTRA_TEXT, "Menu");
194             startService(intent);
195         }
196     });
197
198     @Override
199     public void onOptionsMenuClosed(Menu menu) {
200         // Nothing else to do, closing the Activity.
201         finish();
202     }
203
204     /**
205      * Posts a Runnable at the end of the message loop, overridable for testing.
206      */
207     protected void post(Runnable runnable) {
208         mHandler.post(runnable);
209     }
210
211     /**
212      * The function handle the request to view the beating plot
213      */
214     private void handleViewBeating() {
215         Toast.makeText(this, "Beating", Toast.LENGTH_SHORT).show();
216         // launch a new thread for starting a new live card
217         mHandler.post(new Runnable() {
218             @Override
219             public void run() {
220                 Intent intent = new Intent(MenuActivity.this,
221                     StopwatchService.class);
222                 intent.putExtra(Intent.EXTRA_TEXT, "Beating");
223                 startService(intent);
224             }
225         });
226     }
227
228     /**
229      * The function handle the request to toggle the electrovalve number 8
230      */
231     private void handleToggleEV8() {
232         Toast.makeText(this, "Toggle EV8", Toast.LENGTH_SHORT).show();
233         mHandler.post(new Runnable() {
234             @Override
235             public void run() {
```

```

236         Intent intent = new Intent(MenuActivity.this, HTTPService.class);
237         intent.putExtra("METHOD", "POST");
238         if(Electrovalves.getElectrovalveStatus(7)) {
239             intent.putExtra("LINK",
240                             Electrovalves.URL_ELECTROVALVES_POST_BASE);
241             intent.putExtra("PARAM", "name=EV8&status=off");
242             Electrovalves.setElectrovalvesStatus(7, false);
243         }
244         else {
245             intent.putExtra("LINK",
246                             Electrovalves.URL_ELECTROVALVES_POST_BASE);
247             intent.putExtra("PARAM", "name=EV8&status=on");
248             Electrovalves.setElectrovalvesStatus(7, true);
249         }
250         startService(intent);
251     }
252 }

254 }

256 /**
257 * The function handle the request to toggle the electrovalve number 7
258 */
259 private void handleToggleEV7() {
260     Toast.makeText(this, "Toggle EV7", Toast.LENGTH_SHORT).show();
261     mHandler.post(new Runnable() {
262         @Override
263         public void run() {
264             Intent intent = new Intent(MenuActivity.this, HTTPService.class);
265             intent.putExtra("METHOD", "POST");
266             if(Electrovalves.getElectrovalveStatus(6)) {
267                 intent.putExtra("LINK",
268                             Electrovalves.URL_ELECTROVALVES_POST_BASE);
269                 intent.putExtra("PARAM", "name=EV7&status=off");
270                 Electrovalves.setElectrovalvesStatus(6, false);
271             }
272             else {
273                 intent.putExtra("LINK",
274                             Electrovalves.URL_ELECTROVALVES_POST_BASE);
275                 intent.putExtra("PARAM", "name=EV7&status=on");
276                 Electrovalves.setElectrovalvesStatus(6, true);
277             }
278             startService(intent);
279         }
280     });
281 }

282 /**
283 * The function handle the request to toggle the electrovalve number 6
284 */

```

```

286     */
287
288     private void handleToggleEV6() {
289         Toast.makeText(this, "Toggle EV6", Toast.LENGTH_SHORT).show();
290         mHandler.post(new Runnable() {
291             @Override
292             public void run() {
293                 Intent intent = new Intent(MenuActivity.this, HTTPService.class);
294                 intent.putExtra("METHOD", "POST");
295                 if(Electrovalves.getElectrovalveStatus(5)) {
296                     intent.putExtra("LINK",
297                         Electrovalves.URL_ELECTROVALVES_POST_BASE);
298                     intent.putExtra("PARAM", "name=EV6&status=off");
299                     Electrovalves.setElectrovalvesStatus(5, false);
300                 }
301                 else {
302                     intent.putExtra("LINK",
303                         Electrovalves.URL_ELECTROVALVES_POST_BASE);
304                     intent.putExtra("PARAM", "name=EV6&status=on");
305                     Electrovalves.setElectrovalvesStatus(5, true);
306                 }
307                 startService(intent);
308             }
309         });
310     }
311
312 /**
313 * The function handle the request to toggle the electrovalve number 5
314 */
315
316     private void handleToggleEV5() {
317         Toast.makeText(this, "Toggle EV5", Toast.LENGTH_SHORT).show();
318         mHandler.post(new Runnable() {
319             @Override
320             public void run() {
321                 Intent intent = new Intent(MenuActivity.this, HTTPService.class);
322                 intent.putExtra("METHOD", "POST");
323                 if(Electrovalves.getElectrovalveStatus(4)) {
324                     intent.putExtra("LINK",
325                         Electrovalves.URL_ELECTROVALVES_POST_BASE);
326                     intent.putExtra("PARAM", "name=EV5&status=off");
327                     Electrovalves.setElectrovalvesStatus(4, false);
328                 }
329                 else {
330                     intent.putExtra("LINK",
331                         Electrovalves.URL_ELECTROVALVES_POST_BASE);
332                     intent.putExtra("PARAM", "name=EV5&status=on");
333                     Electrovalves.setElectrovalvesStatus(4, true);
334                 }
335                 startService(intent);
336             }
337         });
338     }

```

```

334         });
335     }
336
337     /**
338      * The function handle the request to toggle the electrovalve number 4
339      */
340     private void handleToggleEV4() {
341         Toast.makeText(this, "Toggle EV4", Toast.LENGTH_SHORT).show();
342         mHandler.post(new Runnable() {
343             @Override
344             public void run() {
345                 Intent intent = new Intent(MenuActivity.this, HTTPService.class);
346                 intent.putExtra("METHOD", "POST");
347                 if(Electrovalves.getElectrovalveStatus(3)) {
348                     intent.putExtra("LINK",
349                         Electrovalves.URL_ELECTROVALVES_POST_BASE);
350                     intent.putExtra("PARAM", "name=EV4&status=off");
351                     Electrovalves.setElectrovalvesStatus(3, false);
352                 }
353                 else {
354                     intent.putExtra("LINK",
355                         Electrovalves.URL_ELECTROVALVES_POST_BASE);
356                     intent.putExtra("PARAM", "name=EV4&status=on");
357                     Electrovalves.setElectrovalvesStatus(3, true);
358                 }
359                 startService(intent);
360             }
361         });
362     }
363
364     /**
365      * The function handle the request to toggle the electrovalve number 3
366      */
367     private void handleToggleEV3() {
368         Toast.makeText(this, "Toggle EV3", Toast.LENGTH_SHORT).show();
369         mHandler.post(new Runnable() {
370             @Override
371             public void run() {
372                 Intent intent = new Intent(MenuActivity.this, HTTPService.class);
373                 intent.putExtra("METHOD", "POST");
374                 if(Electrovalves.getElectrovalveStatus(2)) {
375                     intent.putExtra("LINK",
376                         Electrovalves.URL_ELECTROVALVES_POST_BASE);
377                     intent.putExtra("PARAM", "name=EV3&status=off");
378                     Electrovalves.setElectrovalvesStatus(2, false);
379                 }
380                 else {
381                     intent.putExtra("LINK",
382                         Electrovalves.URL_ELECTROVALVES_POST_BASE);
383                 }
384             }
385         });
386     }

```

```

384         intent.putExtra("PARAM", "name=EV3&status=on");
385         Electrovalves.setElectrovalvesStatus(2, true);
386     }
387     startService(intent);
388 });
389 }
390
391 /**
392 * The function handle the request to toggle the electrovalve number 2
393 */
394 private void handleToggleEV2() {
395     Toast.makeText(this, "Toggle EV2", Toast.LENGTH_SHORT).show();
396     mHandler.post(new Runnable() {
397         @Override
398         public void run() {
399             Intent intent = new Intent(MenuActivity.this, HTTService.class);
400             intent.putExtra("METHOD", "POST");
401             if(Electrovalves.getElectrovalveStatus(1)) {
402                 intent.putExtra("LINK",
403                     Electrovalves.URL_ELECTROVALVES_POST_BASE);
404                 intent.putExtra("PARAM", "name=EV2&status=off");
405                 Electrovalves.setElectrovalvesStatus(1, false);
406             }
407             else {
408                 intent.putExtra("LINK",
409                     Electrovalves.URL_ELECTROVALVES_POST_BASE);
410                 intent.putExtra("PARAM", "name=EV2&status=on");
411                 Electrovalves.setElectrovalvesStatus(1, true);
412             }
413             startService(intent);
414         }
415     });
416 }
417
418 /**
419 * The function handle the request to toggle the electrovalve number 1
420 */
421 private void handleToggleEV1() {
422     Toast.makeText(this, "Toggle EV1", Toast.LENGTH_SHORT).show();
423     mHandler.post(new Runnable() {
424         @Override
425         public void run() {
426             Intent intent = new Intent(MenuActivity.this, HTTService.class);
427             intent.putExtra("METHOD", "POST");
428             if(Electrovalves.getElectrovalveStatus(0)) {
429                 intent.putExtra("LINK",
430                     Electrovalves.URL_ELECTROVALVES_POST_BASE);
431                 intent.putExtra("PARAM", "name=EV1&status=off");
432             }
433         }
434     });
435 }

```

```
432             Electrovalves.setElectrovalvesStatus(0, false);
433         }
434     } else {
435         intent.putExtra("LINK",
436                         Electrovalves.URL_ELECTROVALVES_POST_BASE);
437         intent.putExtra("PARAM", "name=EV1&status=on");
438         Electrovalves.setElectrovalvesStatus(0, true);
439     }
440     startService(intent);
441 }
442 });
443 }
444 /**
445 * The function handle the request to show the Drive Electrovalves card
446 */
447 private void handleViewElectrovalves() {
448     Toast.makeText(this, "Electrovalves", Toast.LENGTH_SHORT).show();
449     mHandler.post(new Runnable() {
450         @Override
451         public void run() {
452             Log.i(LIVE_CARD_TAG, "State = Electrovalves");
453             AppManager.getInstance().setState(ELECTROVALVES);
454         }
455     });
456 }
457 }
458
459 private static void setOptionsMenuState(Menu menu, int menuItemId,
460                                         boolean enabled){
461     MenuItem menuItem = menu.findItem(menuItemId);
462     menuItem.setVisible(enabled);
463     menuItem.setEnabled(enabled);
464 }
```

Listing A.25: AppManager.java

```
1 package com.google.android.glass.sample.klabinterface;  
2  
3 /**  
 * Created by alik on 12/2/2014.  
 */  
5  
public class AppManager {  
7  
    private int state;  
9  
    private static AppManager instance = new AppManager();  
11  
    public static AppManager getInstance(){
```

```
13     return instance;
14 }
15
16     public void setState(int value){
17         state = value;
18     }
19
20     public int getState(){
21         return state;
22     }
23 }
```

Listing A.26: ElectrovalvesView.java

```
package com.google.android.glass.sample.stopwatch;
1
2
3 import android.content.Context;
4 import android.graphics.Color;
5 import android.os.Handler;
6 import android.util.AttributeSet;
7 import android.view.LayoutInflater;
8 import android.widget.RelativeLayout;
9 import android.widget.TextView;
10
11 /**
12 * Subclass of RelativeLayout that is in charge to render the UI of live
13 * card when the user wants to drive the valves.
14 */
15
16 public class ElectrovalvesView extends RelativeLayout {
17
18     private Listener mListener;
19
20     private boolean mRunning = false;
21
22     /**
23      * Array of TextView for each electrovalve
24      */
25
26     private final TextView[] electrovalveText = new TextView[8];
27
28     /**
29      * Interface to listen for changes on the view layout.
30      */
31
32     public interface Listener {
33
34         /**
35          * Notified of a change in the view. */
36         public void onChange();
37     }
38
39     /**
40      * Time delimiter specifying when the second component is fully shown. */
41     private static final long DELAY_MILLIS = 40;
```

```
36     private final Handler mHandler = new Handler();
37     private final Runnable mUpdateViewRunnable = new Runnable() {
38
39         @Override
40         public void run() {
41             if (mRunning) {
42                 updateView();
43                 postDelayed(mUpdateViewRunnable, DELAY_MILLIS);
44             }
45         }
46     };
47
48     public ElectrovalvesView(Context context) {
49         this(context, null);
50     }
51
52     public ElectrovalvesView(Context context, AttributeSet attrs) {
53         this(context, attrs, 0);
54     }
55
56     public ElectrovalvesView(Context context, AttributeSet attrs, int defStyle) {
57         super(context, attrs, defStyle);
58         LayoutInflater.from(context).inflate(R.layout.electrovalves_layout, this);
59
60         electrovalveText[0] = (TextView) findViewById(R.id.ev1);
61         electrovalveText[1] = (TextView) findViewById(R.id.ev2);
62         electrovalveText[2] = (TextView) findViewById(R.id.ev3);
63         electrovalveText[3] = (TextView) findViewById(R.id.ev4);
64         electrovalveText[4] = (TextView) findViewById(R.id.ev5);
65         electrovalveText[5] = (TextView) findViewById(R.id.ev6);
66         electrovalveText[6] = (TextView) findViewById(R.id.ev7);
67         electrovalveText[7] = (TextView) findViewById(R.id.ev8);
68     }
69
70     /**
71      * Sets a Listener.
72      */
73     public void setListener(Listener listener) {
74         mListener = listener;
75     }
76
77     /**
78      * Returns the set Listener.
79      */
80     public Listener getListener() {
81         return mListener;
82     }
```

```

84     @Override
85     public boolean postDelayed(Runnable action, long delayMillis) {
86         return mHandler.postDelayed(action, delayMillis);
87     }
88
89     @Override
90     public boolean removeCallbacks(Runnable action) {
91         mHandler.removeCallbacks(action);
92         return true;
93     }
94
95     /**
96      * Starts the rendering of electrovalves status.
97      */
98     public void start() {
99         if (!mRunning) {
100             postDelayed(mUpdateViewRunnable, DELAY_MILLIS);
101         }
102         mRunning = true;
103     }
104
105     /**
106      * Stops the rendering of electrovalves status.
107      */
108     public void stop() {
109         if (mRunning) {
110             removeCallbacks(mUpdateViewRunnable);
111             // mStarted = false;
112         }
113         mRunning = false;
114     }
115
116     /**
117      * Updates the view to reflect the current state of animation, visible for testing.
118      */
119     void updateView() {
120         if (Electrovalves.isReady()){
121             // electrovalves_status = mElectrovalvesStatus.getElectrovalvesStatus();
122             for(int i = 0; i<Electrovalves.ELECTROVALVES_NUMBER; i++){
123                 if(Electrovalves.getElectrovalveStatus(i)){
124                     electrovalveText[i].setText("EV"+Integer.toString(i+1)+": ON");
125                     electrovalveText[i].setTextColor(Color.GREEN);
126                 }
127                 else{
128                     electrovalveText[i].setText("EV"+Integer.toString(i+1)+": OFF");
129                     electrovalveText[i].setTextColor(Color.RED);
130                 }
131             }
132         }
133     }

```

```

134     if (mListener != null) {
135         mListener.onChange();
136     }
137 }

```

Listing A.27: Electrovalves.java

```

1 package com.google.android.glass.sample.stopwatch;
2
3 import android.util.Log;
4
5 import java.io.OutputStream;
6 import java.io.OutputStreamWriter;
7 import java.lang.Object;
8
9 import org.apache.http.HttpResponse;
10 import org.apache.http.client.HttpClient;
11 import org.apache.http.client.methods.HttpGet;
12 import org.apache.http.impl.client.DefaultHttpClient;
13
14 import java.io.BufferedInputStream;
15 import java.io.BufferedReader;
16 import java.io.ByteArrayOutputStream;
17 import java.io.IOException;
18 import java.io.InputStream;
19 import java.io.InputStreamReader;
20 import java.net.HttpURLConnection;
21 import java.net.MalformedURLException;
22 import java.net.URL;
23 import java.net.URLConnection;
24 import java.util.concurrent.Semaphore;
25
26 /**
27 * Static class where the electrovalves status are memorized
28 */
29 public class Electrovalves {
30
31     private static boolean mReady;
32
33     /** Number of Electrovalves */
34     public static final int ELECTROVALVES_NUMBER = 8;
35
36     /** URL where the electrovalves information is stored */
37     public static final String URL_ELECTROVALVES_BASE =
38                     "http://CENSURED/show/";
39
40     /** URL where the electrovalves information has to be post */
41     public static final String URL_ELECTROVALVES_POST_BASE =

```

```
43     /** URL to set the value of the electrovalves */
44     public static final String URL_SET_ELECTROVALVES_BASE =
45                                         "http://CENSURED/add/EV" ;
46
47     /** Array where the electrovalves status is stored */
48     public static boolean[] electrovalves_status =
49                                         new boolean[ELECTROVALVES_NUMBER];
50
51     public static boolean getElectrovalveStatus(int index)
52     {
53         return electrovalves_status[index];
54     }
55
56     /** Setter for the Electrovalves Status
57      *
58      * @param index , position in the electrovalves_status array
59      * @param value , boolean, true if the electrovalves is on, false otherwise
60      */
61     public static void setElectrovalvesStatus(int index, boolean value){
62         electrovalves_status[index] = value;
63         Log.i("DRAWER", "EV" + Integer.toString(index + 1) + " has been set");
64         if(index == ELECTROVALVES_NUMBER-1) {
65             setReady(true);
66             Log.i("ELECTROVALVES", "status ready");
67         }
68     }
69
70     public static boolean isReady(){
71         return mReady;
72     }
73
74     public static void setReady(boolean value){
75         mReady = value;
76     }
77 }
```

Listing A.28: `HTTPService.java`

```
1 package com.google.android.glass.sample.stopwatch;

3 import android.app.Service;
import android.content.Intent;
5 import android.os.AsyncTask;
import android.os.IBinder;
7 import android.util.Log;

9 import java.io.BufferedInputStream;
import java.io.InputStream;
```

```
11 import java.io.OutputStreamWriter;
12 import java.net.HttpURLConnection;
13 import java.net.URL;
14
15 public class HTTPService extends Service {
16
17     String httpMethod;
18     String httpLink;
19     HttpURLConnection mUrlConnection;
20     String mResult;
21     String urlParameters;
22
23     public HTTPService() {
24 }
25
26     @Override
27     public IBinder onBind(Intent intent) {
28         // TODO: Return the communication channel to the service.
29         throw new UnsupportedOperationException("Not yet implemented");
30     }
31
32     @Override
33     public int onStartCommand(Intent intent, int flags, int startId) {
34         //from the intent I take which http method is it
35         httpMethod = intent.getStringExtra("METHOD");
36         Log.i("HTTP METHOD", httpMethod);
37         httpLink = intent.getStringExtra("LINK");
38         Log.i("HTTP LINK", httpLink);
39         urlParameters = intent.getStringExtra("PARAM");
40         Log.i("HTTP PARAM", urlParameters);
41         new HttpRequest().execute();
42
43         return START_NOT_STICKY;
44     }
45
46     // Async task class to make HTTP GET and POST
47     private class HttpRequest extends AsyncTask<Void, Void, Void> {
48         @Override
49         protected Void doInBackground(Void... arg0){
50             try{
51                 if (httpMethod.equalsIgnoreCase("GET")){
52                     URL url = new URL(httpLink);
53                     mUrlConnection = (HttpURLConnection) url.openConnection();
54                     InputStream in = new BufferedInputStream(
55                         mUrlConnection.getInputStream());
56                     int ch;
57                     StringBuffer b = new StringBuffer();
58                     while ((ch = in.read()) != -1){
59                         b.append((char) ch);
60                     }
61                     mResult = b.toString();
62                 }
63             } catch (Exception e) {
64                 e.printStackTrace();
65             }
66         }
67         @Override
68         protected void onPostExecute(Void result) {
69             super.onPostExecute(result);
70             Intent intent = new Intent("com.example.intent.action.RESULT");
71             intent.putExtra("RESULT", mResult);
72             LocalBroadcastManager.getInstance(getApplicationContext())
73                 .sendBroadcast(intent);
74         }
75     }
76 }
```

```
61         }
62         mResult = new String(b);
63     }
64
65     if (httpMethod.equalsIgnoreCase("POST")) {
66
67         URL url = new URL(httpLink);
68         mURLConnection = (HttpURLConnection) url.openConnection();
69         mURLConnection.setRequestMethod("POST");
70
71         OutputStreamWriter writer = new OutputStreamWriter(
72             mURLConnection.getOutputStream());
73         writer.write(urlParameters);
74         writer.flush();
75
76
77         InputStream in = new BufferedInputStream(
78             mURLConnection.getInputStream());
79         int ch;
80         StringBuffer b = new StringBuffer();
81         while ((ch = in.read()) != -1){
82             b.append((char) ch);
83         }
84         mResult = new String(b);
85         in.close();
86         writer.close();
87     }
88
89 }  
}
```

Listing A.29: DataPoint.java

```
1 package com.google.android.glass.sample.klabinterface;
2
3 /**
4 * Created by Buso on 28/10/2014.
5 */
6 class DataPoint {
7     private final double mTimestamp;
8     private final double mValue;
9     DataPoint(double timestamp, double value) {
10         mTimestamp = timestamp;
11         mValue = value;
12     }
13     public double getTimestamp() {
14         return mTimestamp;
15     }
16 }
```

```
16     public double getValue() {  
17         return mValue;  
18     }  
}
```

[13]

LIST OF FIGURES

Figure 1	<i>XCEL</i> project (<i>Body-on-a-chip</i>)	1
Figure 2	Block diagram of the system	2
Figure 3	Glasswear's Block Diagram	3
Figure 4	Drive Electrovalves Steps	4
Figure 5	<i>Back</i> menu item	4
Figure 6	<i>Exit</i> menu item	4
Figure 7	The Board	5
Figure 8	Storing Microscope Video Program's Icon	6
Figure 9	Storing Microscope Video Console	7
Figure 10	Video Stored in the Folder	7
Figure 11	Multisensor - Size Comparison	11
Figure 12	Multisensor - Scheme	12
Figure 13	PH and Temperature Sensors from an Optical Image	13
Figure 14	Typical pH-sensor transfer function	14
Figure 15	Conditioning circuit for pH sensor	15
Figure 16	Error caused by Amplifier's Input Bias Current	15
Figure 17	Low-Pass Filter Schematic	17
Figure 18	Low-Pass Filter Frequency Response	17
Figure 19	Acquisition Path for pH Sensor	18
Figure 20	Conditioning Circuit for Temperature Sensor	19
Figure 21	Acquisition Path for pH Sensor	20
Figure 22	Electrovalves	21
Figure 23	Driver for electrovalves	21
Figure 24	Electrovalves Driver Circuit	22
Figure 25	DC-DC circuit	23
Figure 26	Relay circuit	24
Figure 27	Electrovalves Supply Circuit	25
Figure 28	Final system mounted on a PCB	26
Figure 29	Schematic of Complete Circuit	27
Figure 30	<i>DC-DC</i> High Speed Switching Path	28
Figure 31	<i>PCB</i> Layout of <i>DC-DC</i>	28
Figure 32	PCB of the System	30
Figure 33	High Level System's Block Diagram	32
Figure 34	Distribution of Internet of Things	33
Figure 35	The <i>IoT</i> Evolution Prediction	34
Figure 36	The <i>Beaglebone Black</i>	34
Figure 37	Embedded Linux Firmware - Blocks Subdivision	36
Figure 38	Bash Script Flow Chart	37
Figure 39	Sensor UML Description	39

Figure 40	Google App Engine Logo	43
Figure 41	Flask Logo	44
Figure 42	Picture Submitting from the Browser	45
Figure 43	Video Submitting from the Brower	46
Figure 44	Video Storing: Block Diagram	48
Figure 45	Video Storing: Sequence Diagram	48
Figure 46	Computing Evolution	50
Figure 47	Google Glass	51
Figure 48	Interactions Between Class	52
Figure 49	MainService's Runnable in Background	52
Figure 50	Live Card, Adding to the Timeline	53
Figure 51	Main Menu	55
Figure 52	Graph Menu	55
Figure 53	Menu Electrovalves	55
Figure 54	Main View	56
Figure 55	Temperature View	56
Figure 56	Beating View	57
Figure 57	<i>Electrovalves View</i>	57
Figure 58	<i>Microscope Video</i>	58
Figure 59	Circuit mounted on breadboard side view	60
Figure 60	Driver for LED	61
Figure 61	LEDs experiments board	61
Figure 62	Circuit mounted on breadboard top view	62

LIST OF LISTINGS

A.1	KlabFirware (bash)	64
A.2	SensorAcquiring (bash)	65
A.3	Pins Setting	65
A.4	ElectrovalvesDriver.cpp	66
A.5	sensor_aquiring.cpp	68
A.6	compute_beating.cpp	71
A.7	Pins Setting	73
A.8	HTTP.h	74
A.9	HTTP.cpp	74
A.10	main.cpp	75
A.11	VideoStoring.pro	76
A.12	mytimer.h	76
A.13	mytimer.cpp	77
A.14	downloader.h	77
A.15	downloader.cpp	78
A.16	Main Script of Server	81
A.17	MainService.java	88
A.18	AppDrawer.java	104
A.19	MainView.java	112
A.20	BeatingView.java	114
A.21	PHViewer.java	117
A.22	TemperatureView.java	120
A.23	VideoPlayerActivity.java	123
A.24	MenuActivity.java	125
A.25	AppManager.java	134
A.26	ElectrovalvesView.java	135
A.27	Electrovalves.java	138
A.28	HTTPService.java	139
A.29	DataPoint.java	141

BIBLIOGRAPHY

- [1] Andrea Cavallini Camilla Baj-Rossi Sara Ghoreishizadeh Giovanni De Micheli Sandro Carrara. Design, fabrication, and test of a sensor array for perspective biosensing in chronic pathologies. *IEEE Biomedical Circuits and System Conference*, 2012. URL http://si2.epfl.ch/~demichel/publications/archive/2012/BioCAS_2012_Andrea.pdf.
- [2] Texas Instruments. Pcb design guidelines for reduced emi. *Application Note*, November 1999. URL <http://www.ti.com/lit/an/szza009/szza009.pdf>.
- [3] Derek Molloy. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. Wiley, 2014. ISBN 1118935128. URL <http://www.exploringbeaglebone.com/>.
- [4] Avi Baum. An-1852 designing with ph electrodes. *Application Note, Texas Instruments*, July 2014. URL <http://www.ti.com/lit/wp/swry009/swry009.pdf>.
- [5] Dave Evans. The internet of things, how the next evolution of the internet is changing everything. *Cisco Systems*, April 2011. URL http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.
- [6] Charles Severance. *Using Google App Engine*. O'Reilly Media, May 2009. ISBN 0596555806.
- [7] Allen Firstenberg Jason Salas. *Designing and Developing for Google Glass: Thinking Differently for a New Platform*. O'Reilly, 2014. ISBN 1491946458.
- [8] Jeff Tang. *Beginning Google Glas Development*. Apress, 2014. ISBN 1430267887.
- [9] Shalabh Aggarwal. *Flask Framework Cookbook*. Packt Publishing, November 2014. ISBN 9781783983407.
- [10] Walt Jung. *Op Amp Applications Handbook*. Newnes, 2005. ISBN 0750678445. URL http://www.analog.com/library/analogDialogue/archives/39-05/op_amp_applications_handbook.html.
- [11] Eric Redmond. *Programming Google Glass, The Mirror API*. The Pragmatic Bookshelf, 2013. ISBN 9781937785796.
- [12] Christian Kueck. Power supply layout and emi. *Application Note, Linear Technology*, October 2012. URL <http://cds.linear.com/docs/en/application-note/an139f.pdf>.
- [13] Texas Instruments. An-1852 designing with ph electrodes. *Application Note*, September 2008–Revised April 2013. URL <http://www.ti.com/lit/an/snoa529a/snoa529a.pdf>.

- [14] Texas Instruments. Lmp7721 3-femtoampere input bias current precision amplifier. *Datasheet*, January 2008–Revised December 2014. URL <http://www.ti.com/lit/ds/symlink/lmp7721.pdf>.
- [15] Texas Instruments. Lm4140 high precision low noise low dropout voltage reference. *Datasheet*, June 2000–Revised April 2013. URL <http://www.ti.com/lit/ds/symlink/lm4140.pdf>.