

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.

In this master thesis will focus on designing of 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 glass-wear¹.

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 ends this thesis, showing the obtained results with different experiments.

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
6 MICROSCOPE VIDEO STORING	45
iv GOOGLE GLASS APPLICATION	46
7 INTRODUCTION	47
v CONCLUSION AND APPENDIX	48
8 TEST AND PERFORMANCE	49
8.1 LED Experiments	49
8.2 Electrovalves experiments	50
8.2.1 Breadboard Phase	50
8.2.2 PCB Phase	52

A CODE	53
A.1 Firmware	53
A.2 Video Storing Software	64
A.3 Google App Engine	69
A.4 Glassware	76
List of Figures	121
List of Listings	122
Bibliography	123

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*.

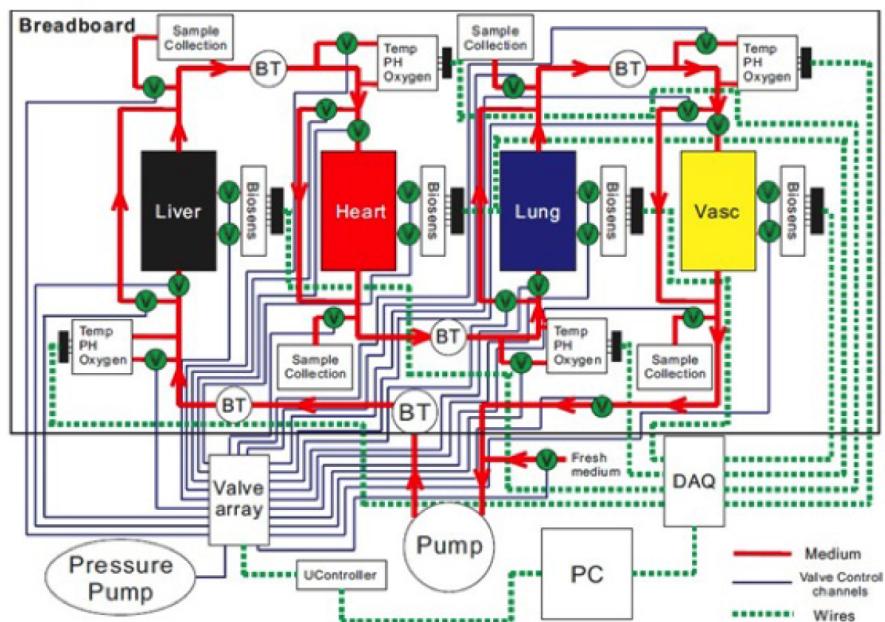


Figure 1: XCEL project (*Body-on-a-chip*)

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

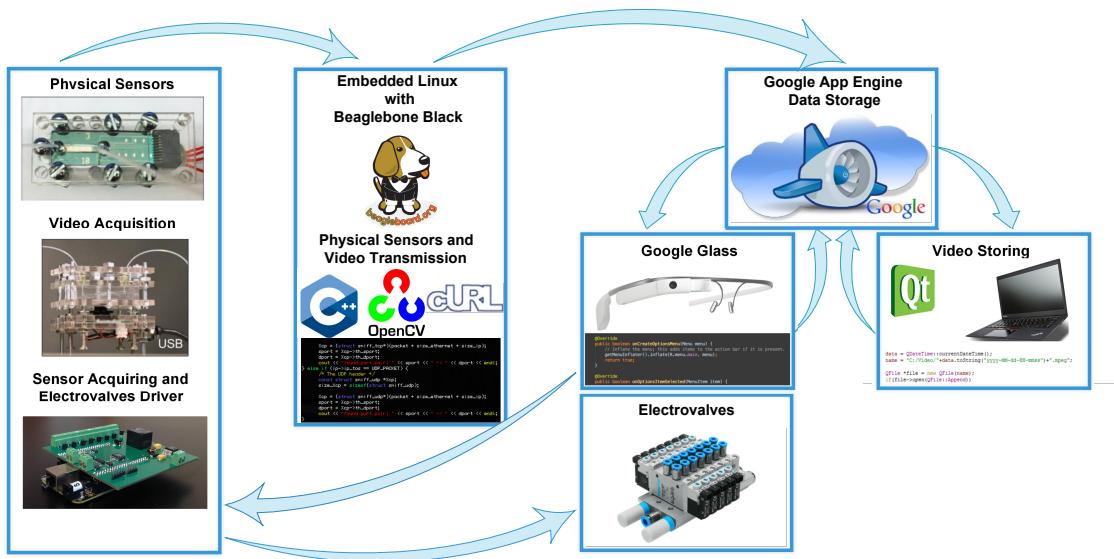


Figure 2: Block diagram of the system

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, 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 electrovalves.

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 in the application (Fig.3c). From this point, tapping and swiping, it's 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.

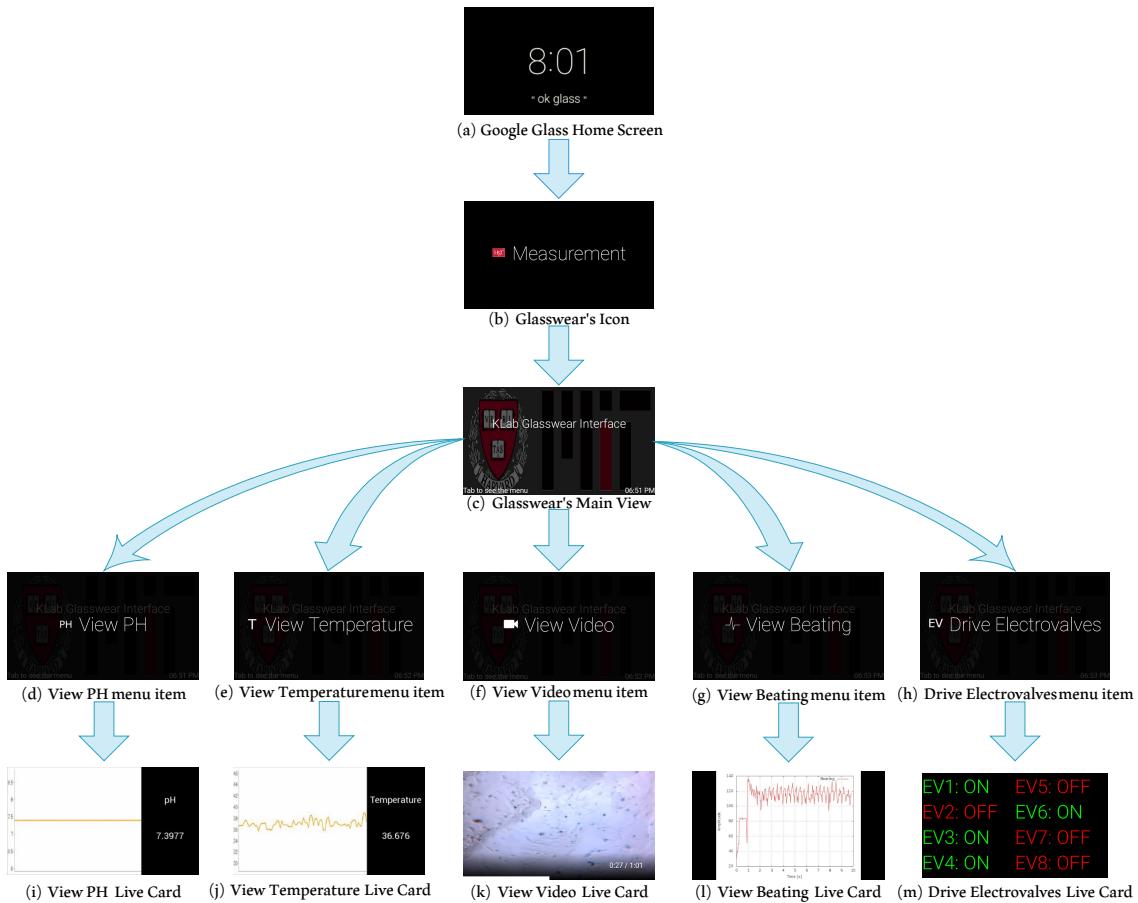


Figure 3: Glasswear's Block Diagram

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.

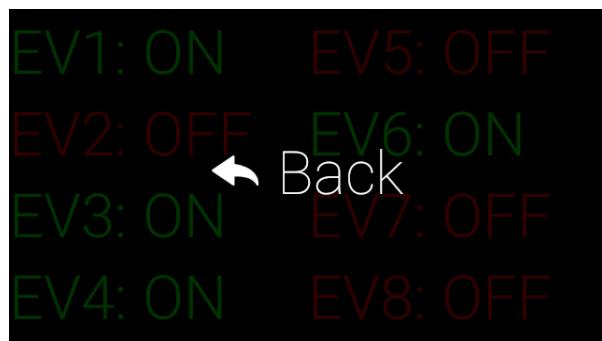


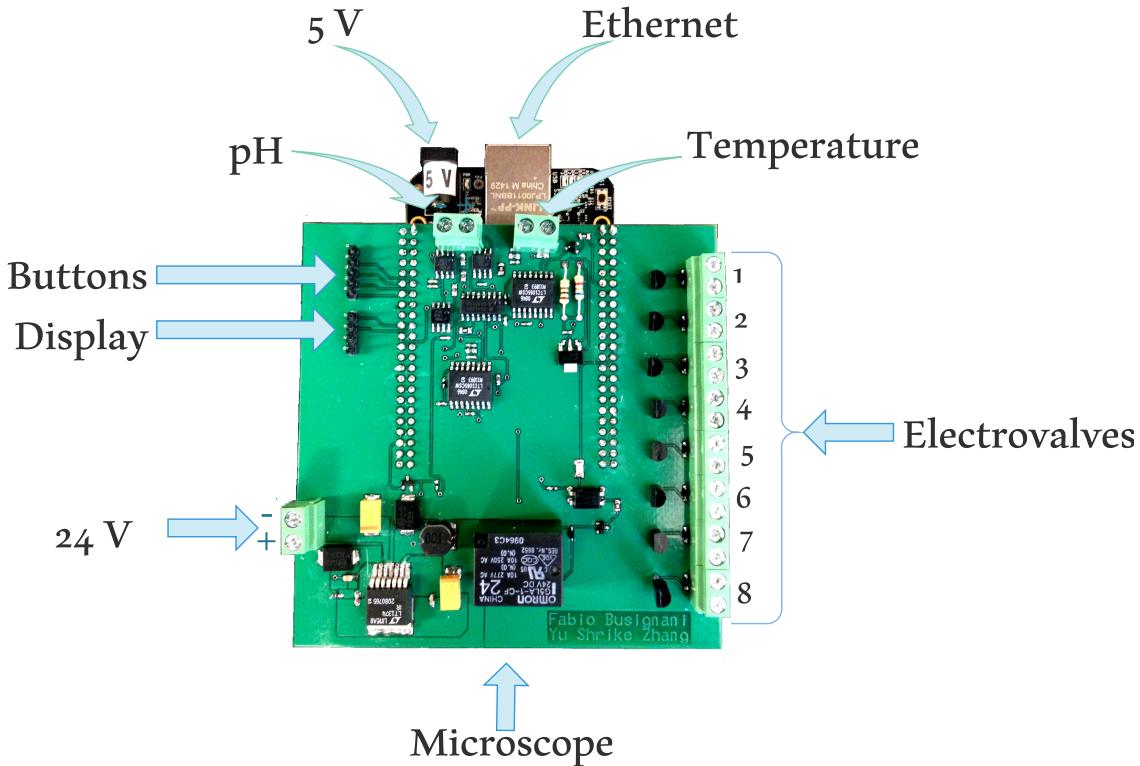
Figure 5: *Back* menu item

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.

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

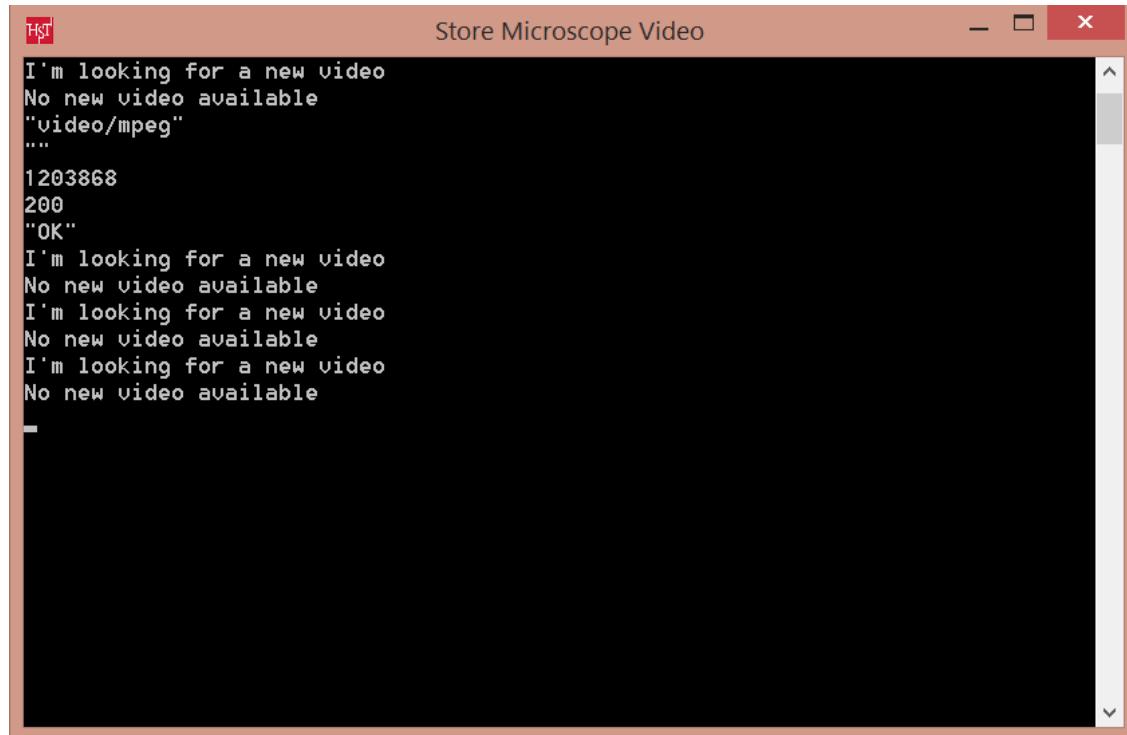
The program is very easy to use, the user just has to run the executable (Fig.8).



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 date 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

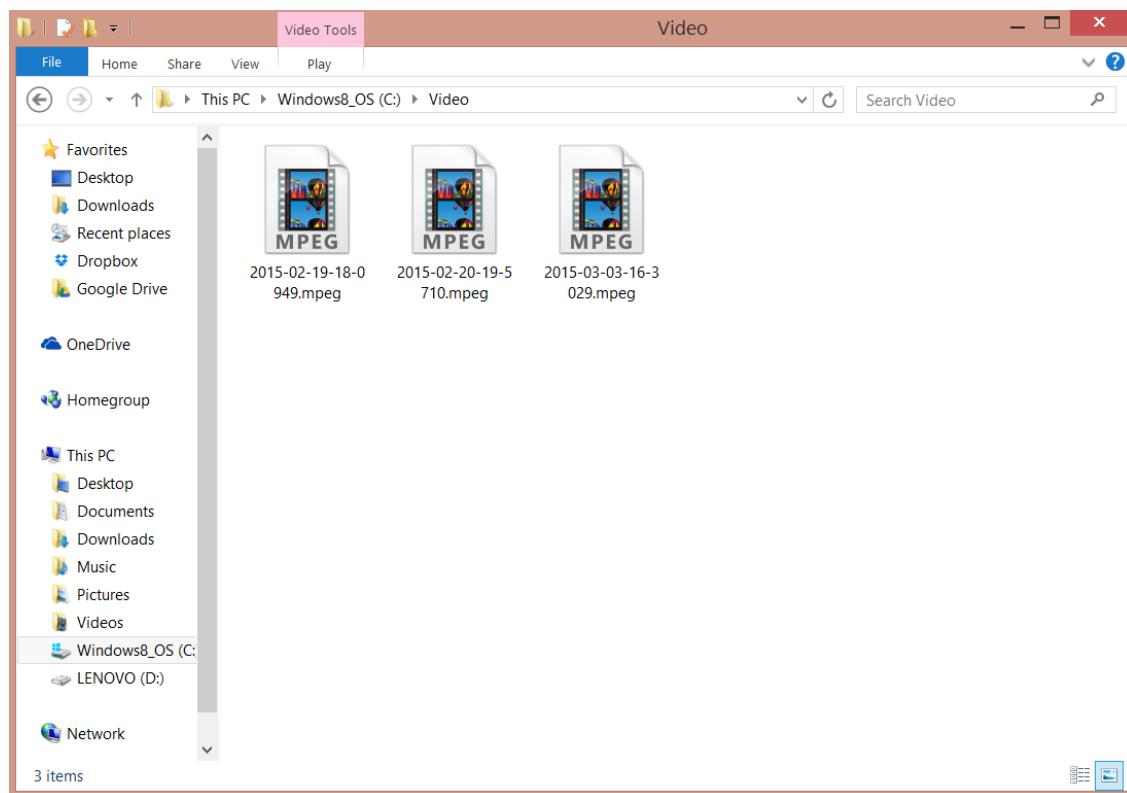


Figure 10: Video Stored in the Folder

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.

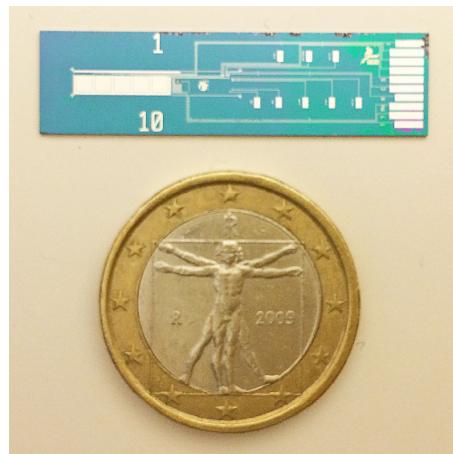


Figure 11: Multisensor - Size Comparison

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.

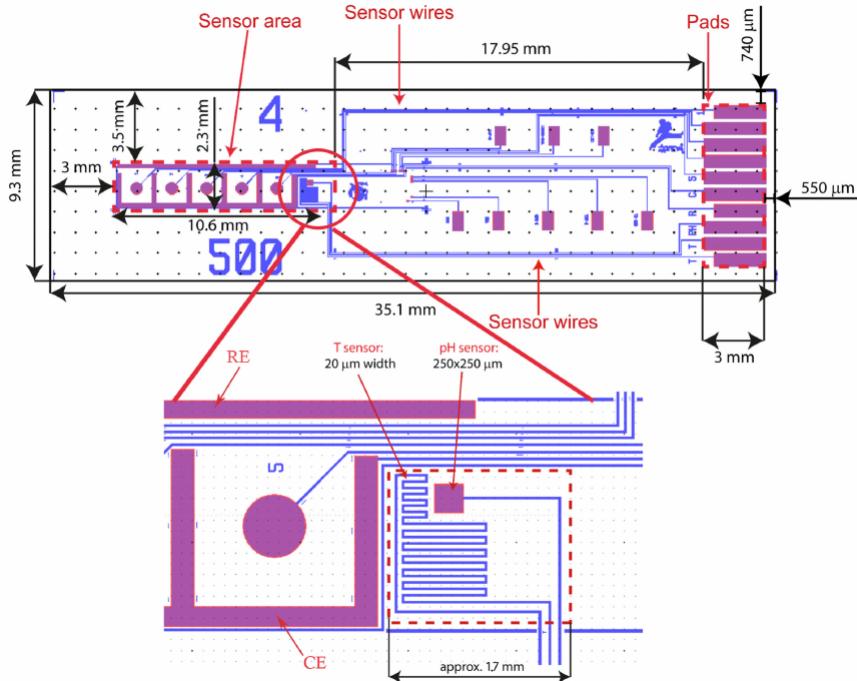


Figure 12: Multisensor - Scheme

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 $\delta = 1.49$.

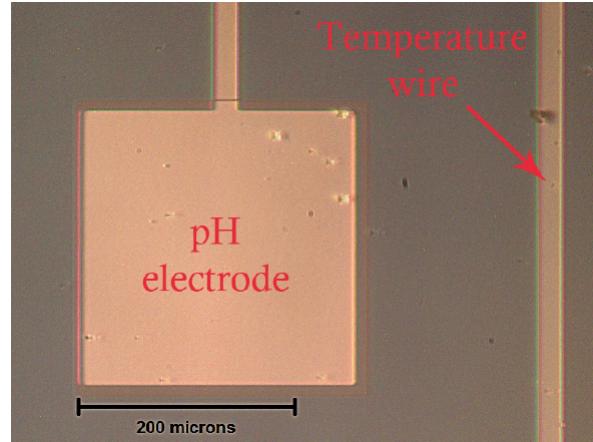


Figure 13: PH and Temperature Sensors from an Optical Image

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).

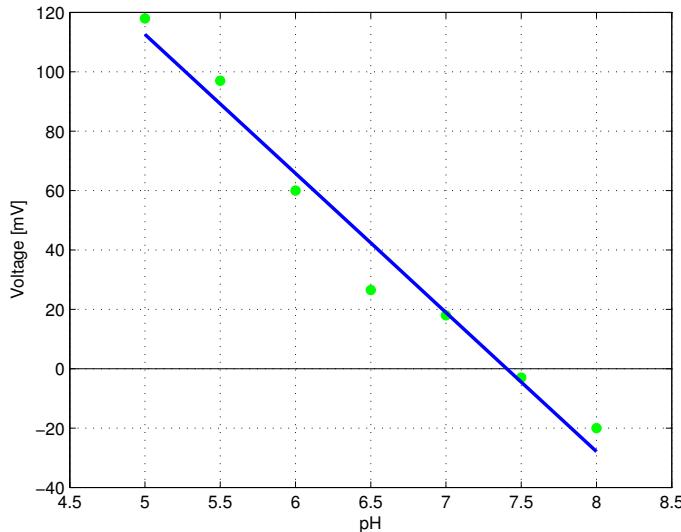


Figure 14: Typical pH-sensor transfer function

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.

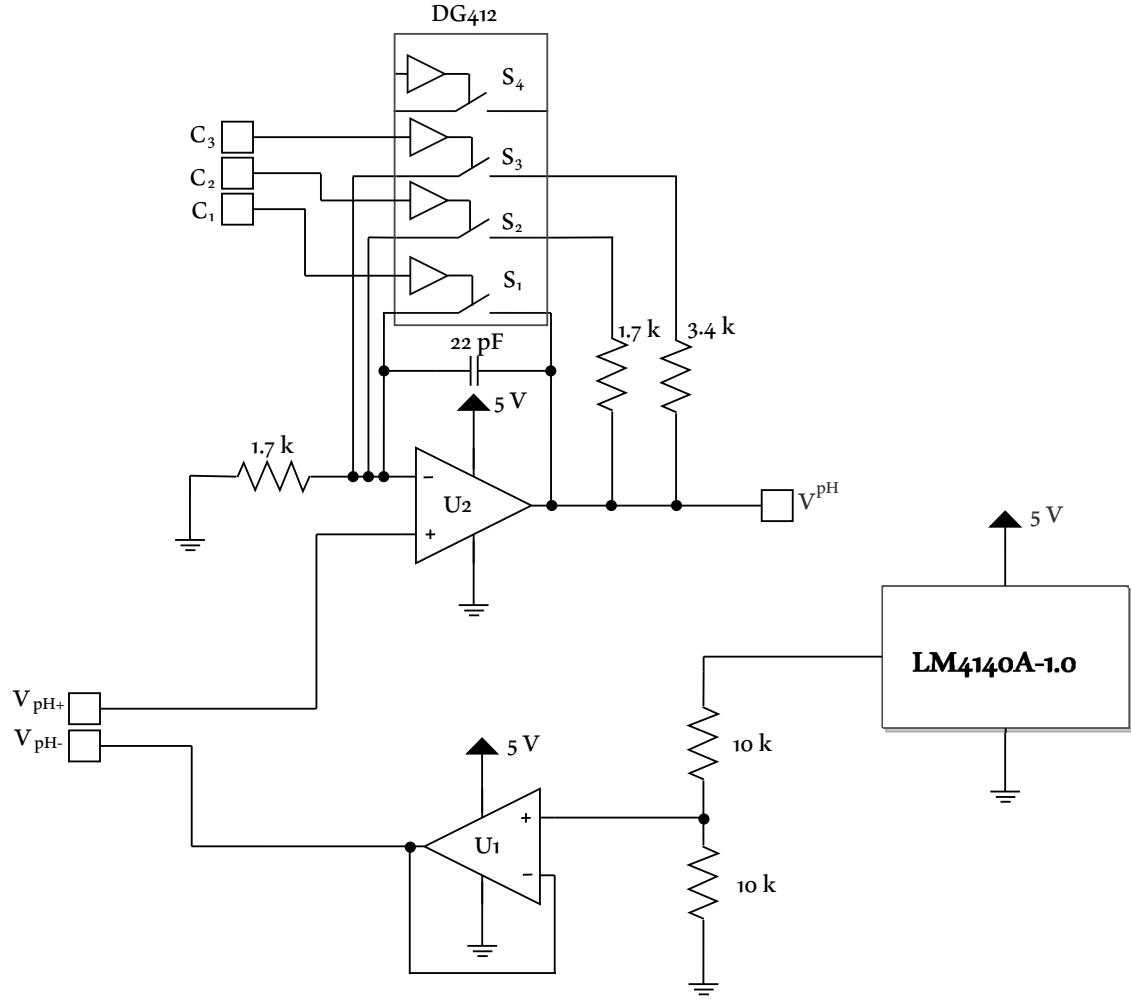


Figure 15: Conditioning circuit for pH sensor

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 ($3 \pm 17 fA$).

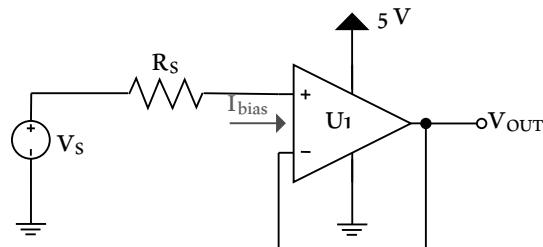


Figure 16: Error caused by Amplifier's Input Bias Current

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 $26 \mu\text{V}$.

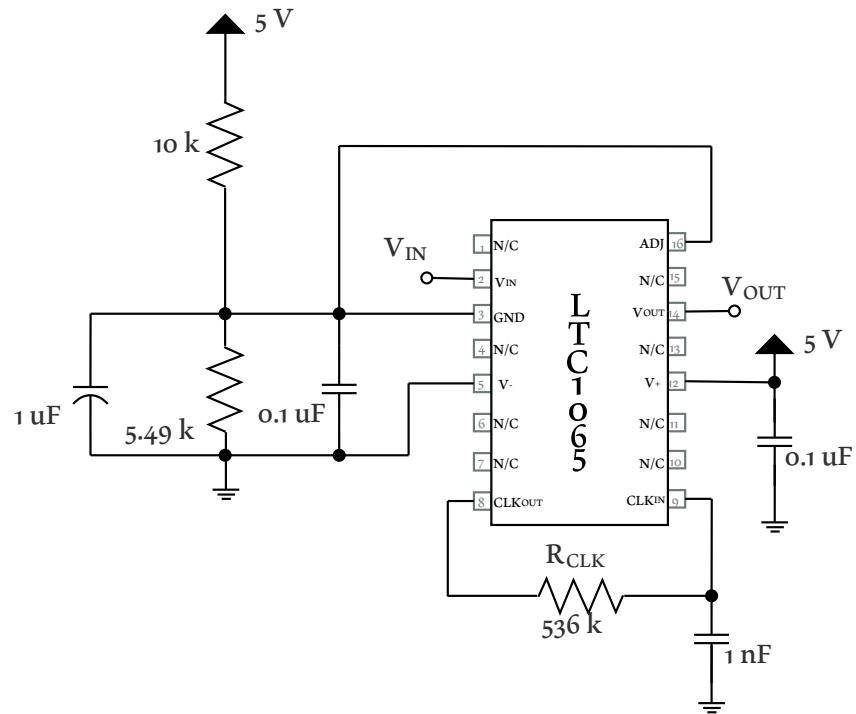
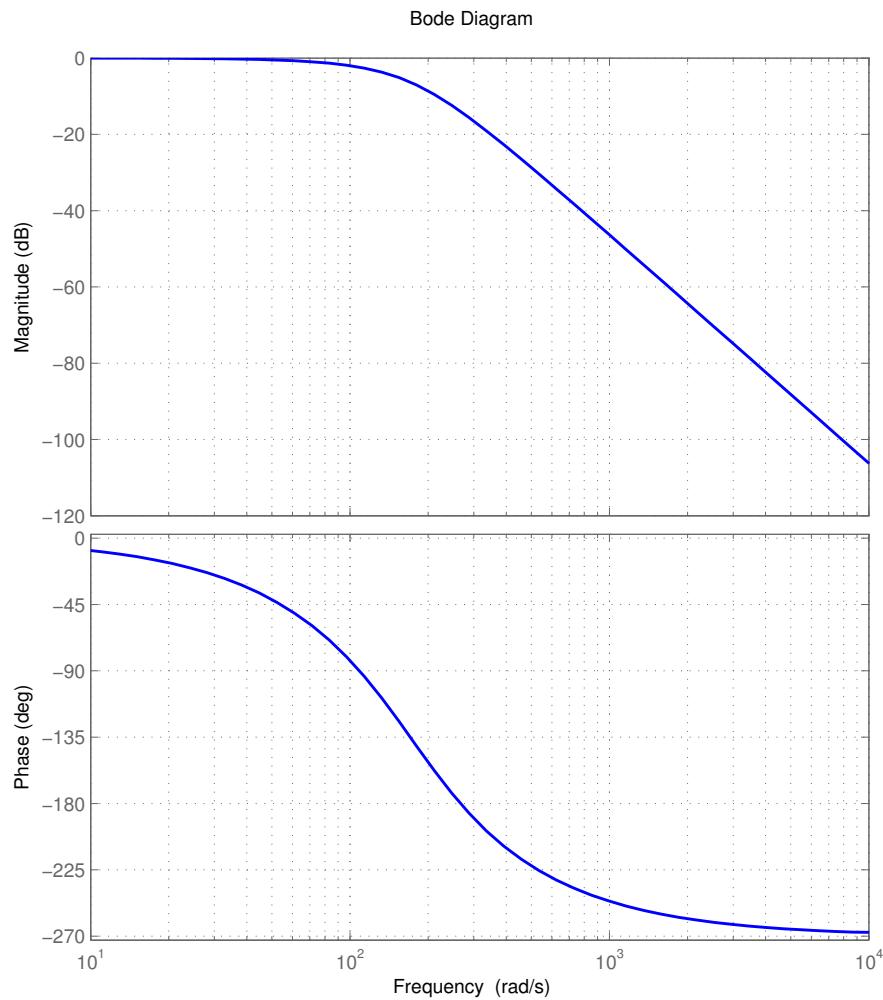
Since the environment in which the circuit is going to be used is a laboratory, so a really noisy place, the conditioning circuit for the sensor has to involve the design of a **low-pass filter** in order to reject the noise.

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.

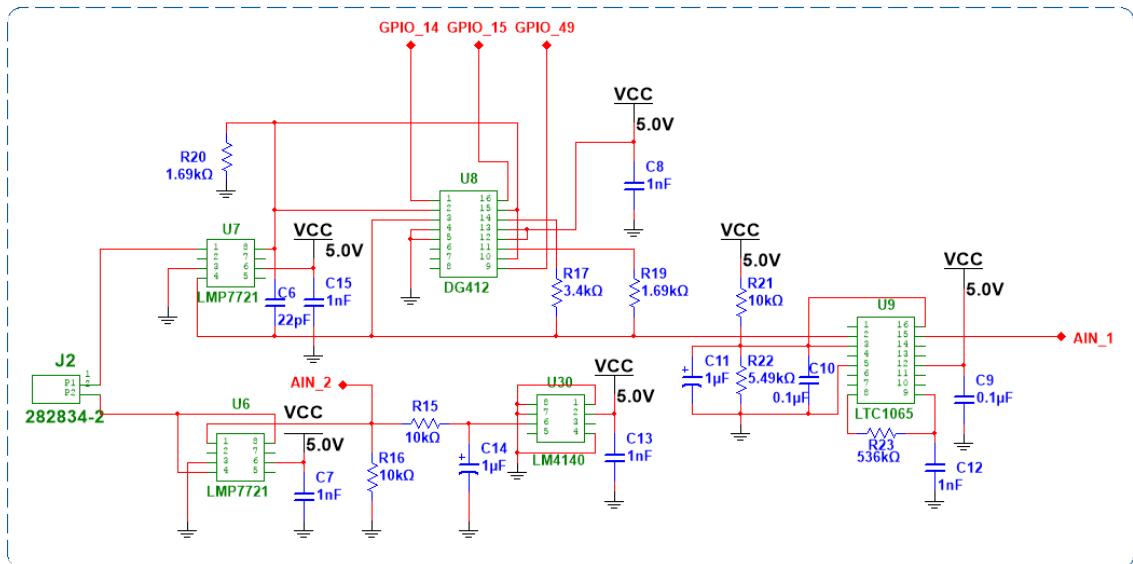


Figure 19: Acquisition Path for pH Sensor

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:

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

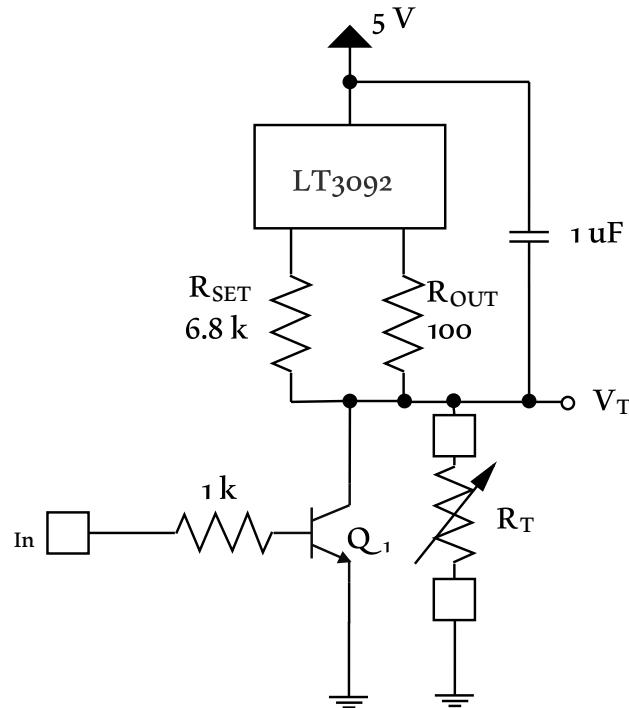


Figure 20: Conditioning Circuit for Temperature Sensor

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.

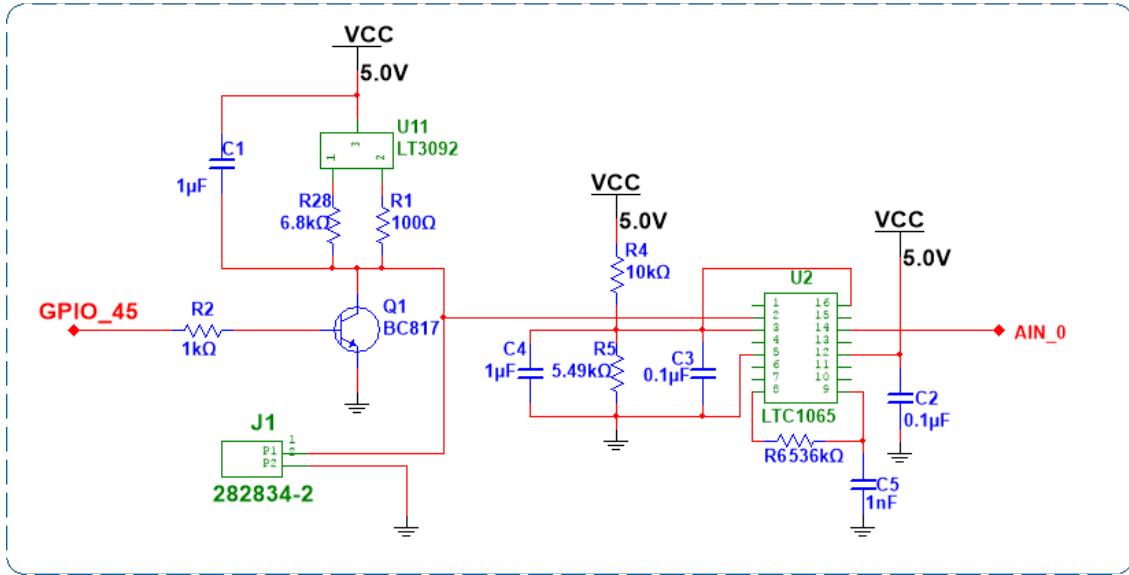


Figure 21: Acquisition Path for pH Sensor

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

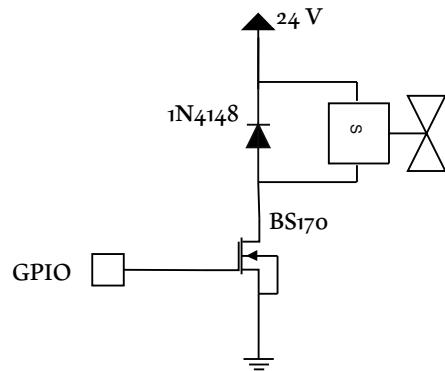


Figure 23: Driver for electrovalves

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.

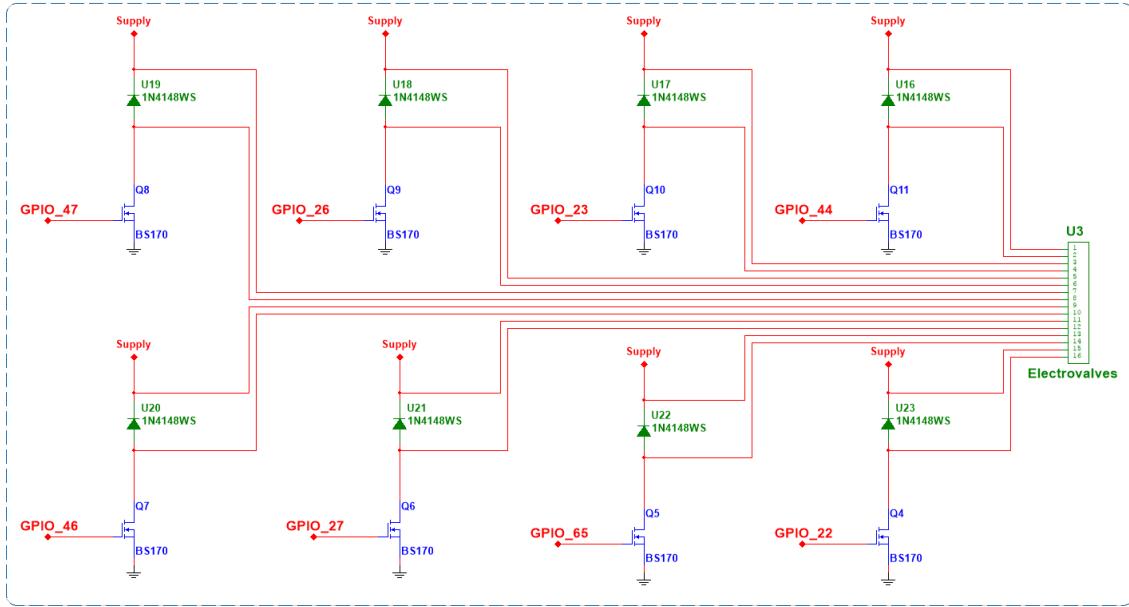


Figure 24: Electrovalves Driver Circuit

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.

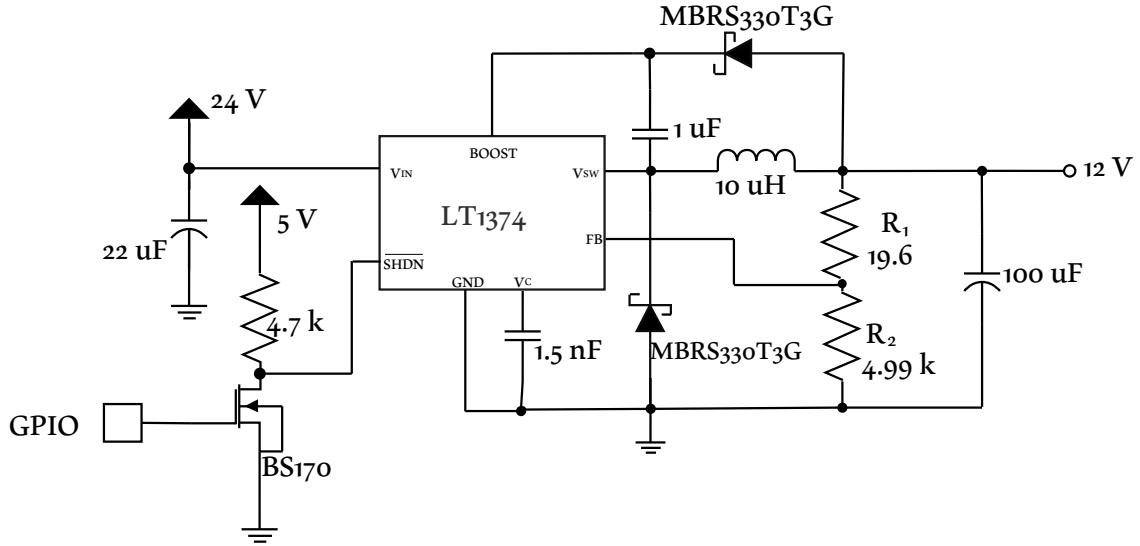


Figure 25: DC-DC circuit

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.

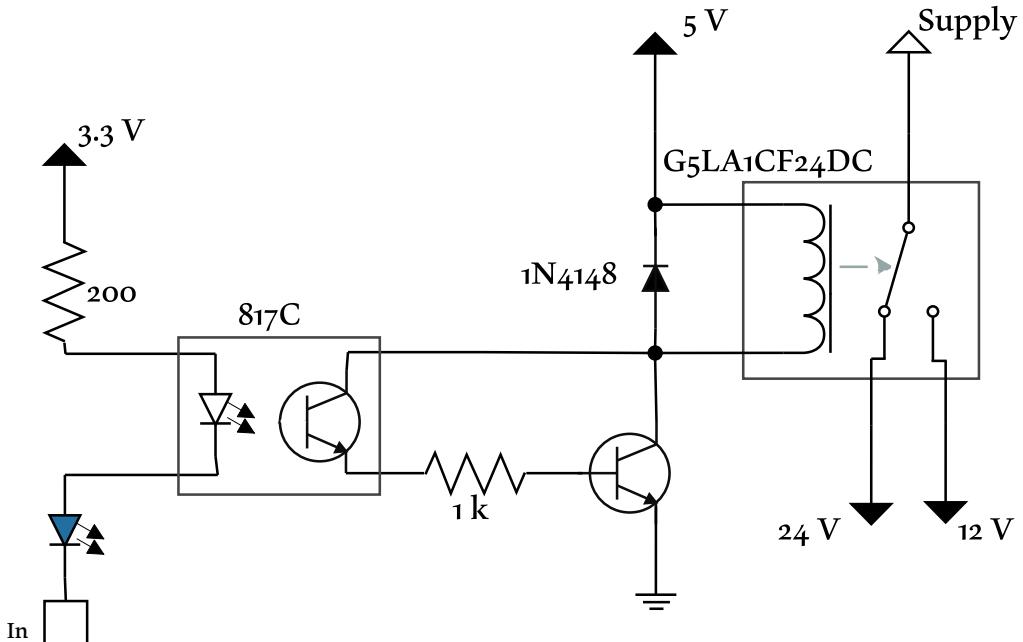


Figure 26: Relay circuit

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 .

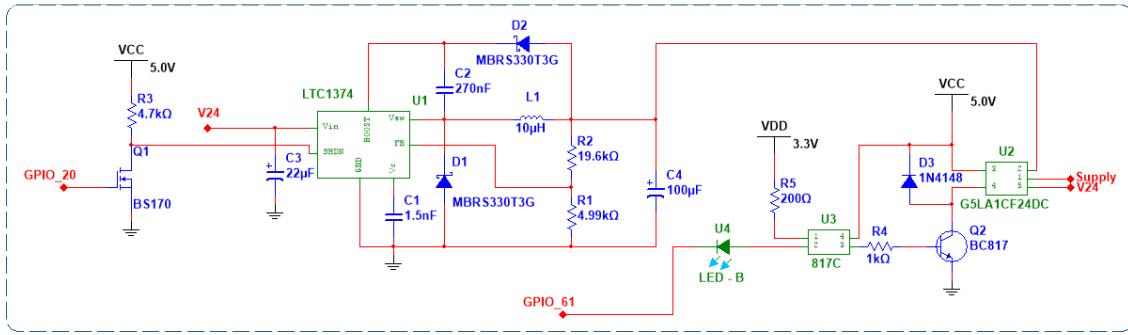


Figure 27: Electrovalves Supply Circuit

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

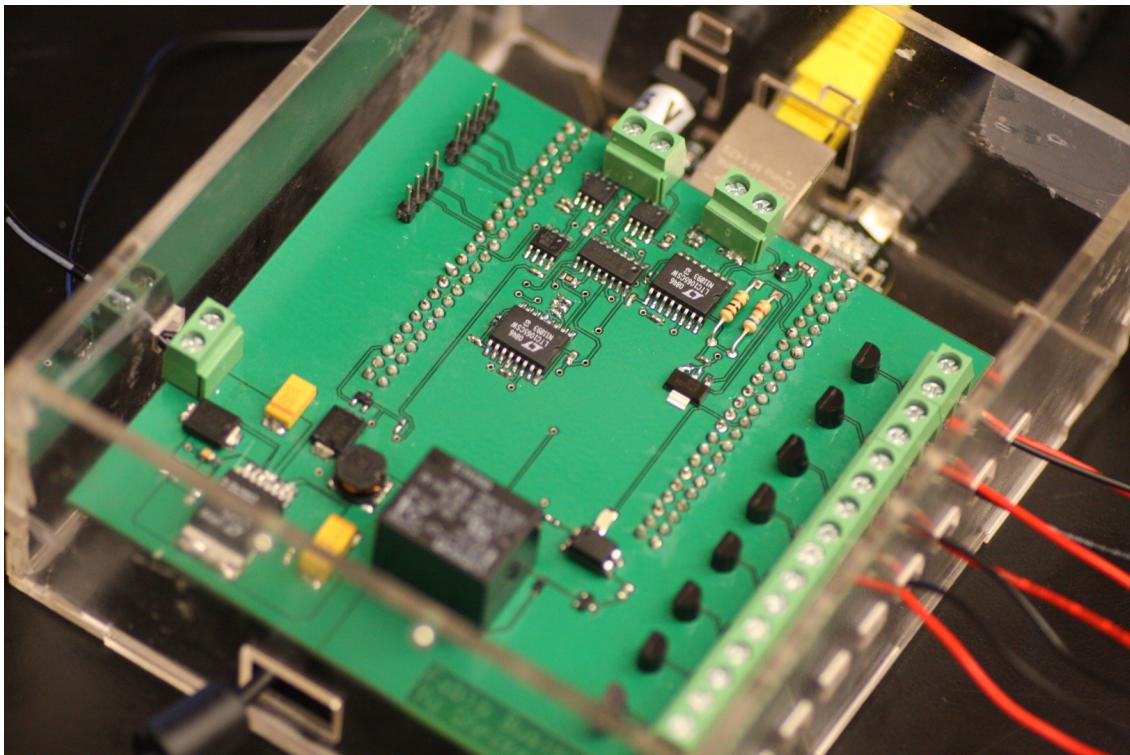


Figure 28: Final system mounted on a PCB

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.

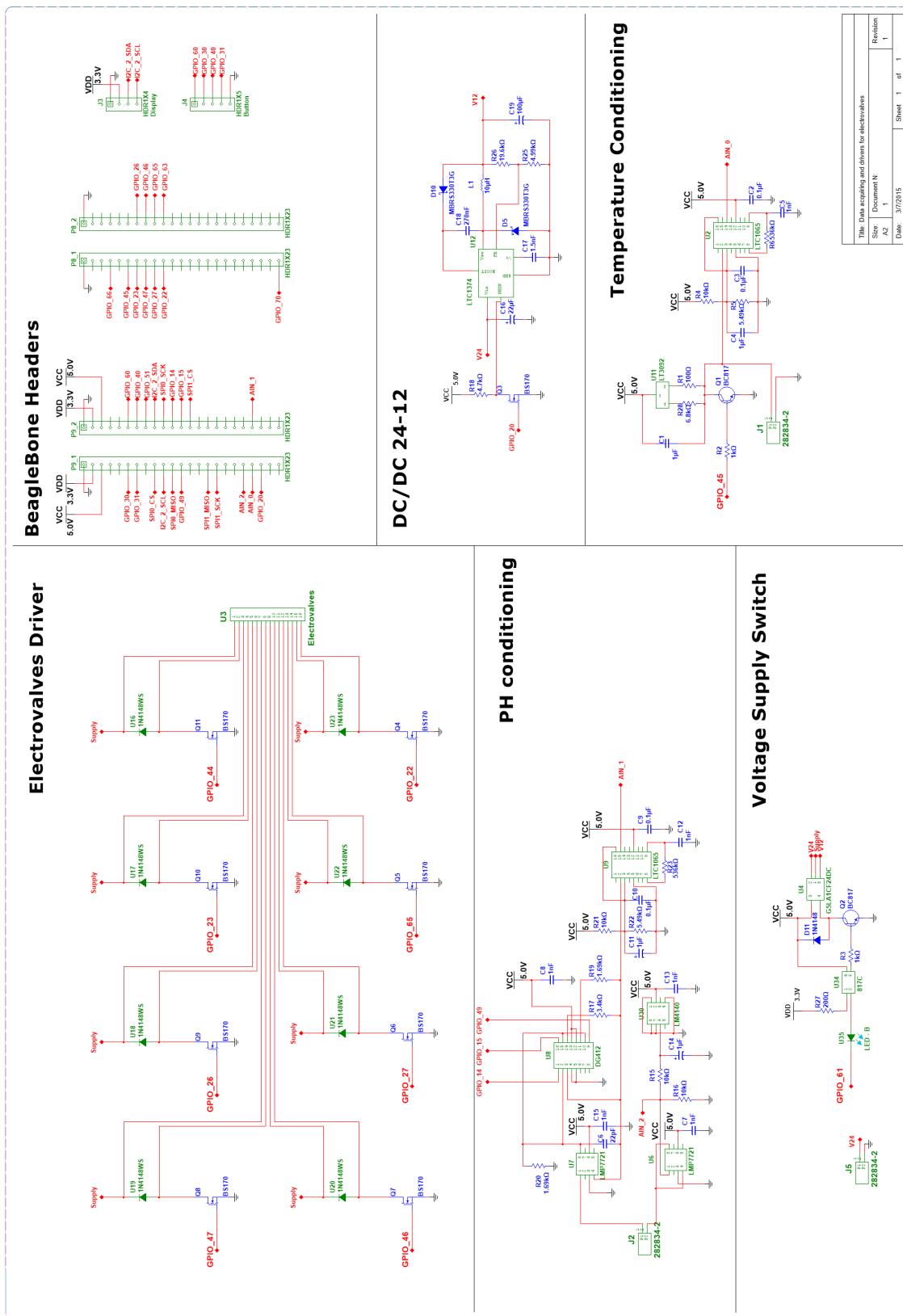


Figure 29: Schematic of Complete Circuit

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**.

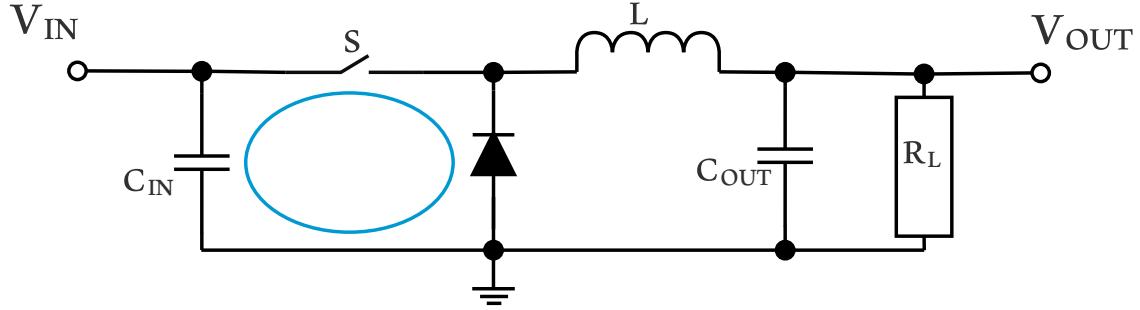


Figure 30: DC-DC High Speed Switching Path

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).

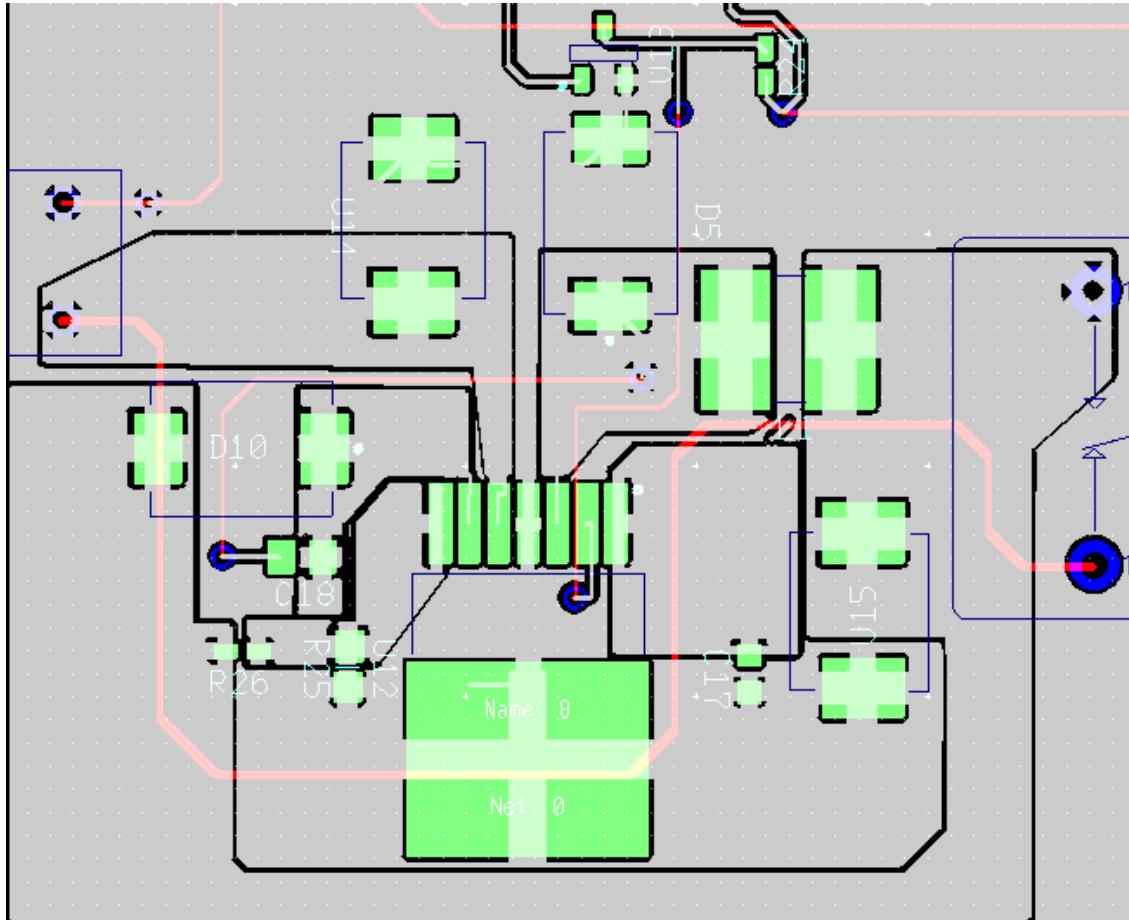


Figure 31: PCB Layout of DC-DC

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.

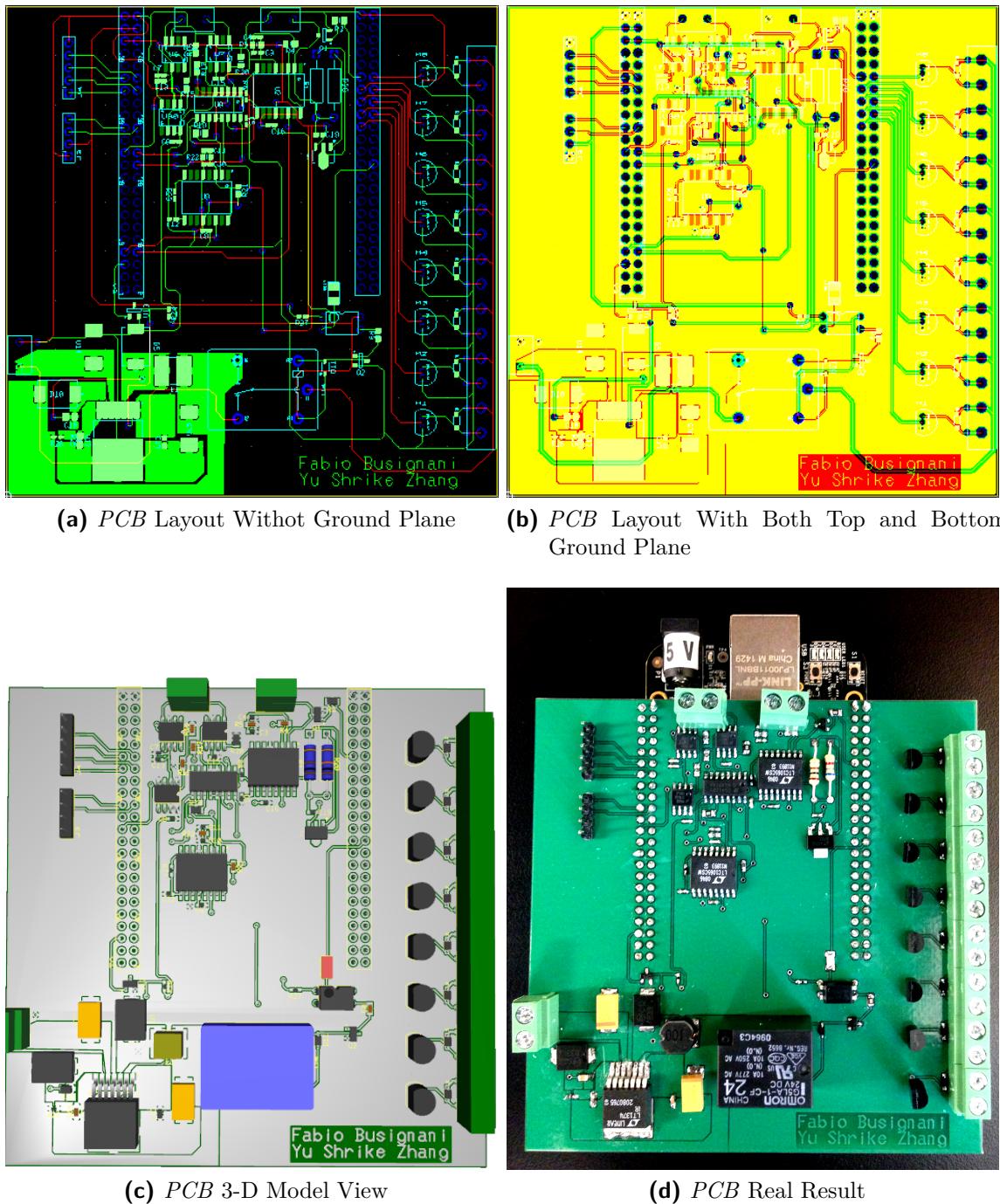


Figure 32: PCB of the System

Part II

FIRMWARE

3

INTRODUCTION

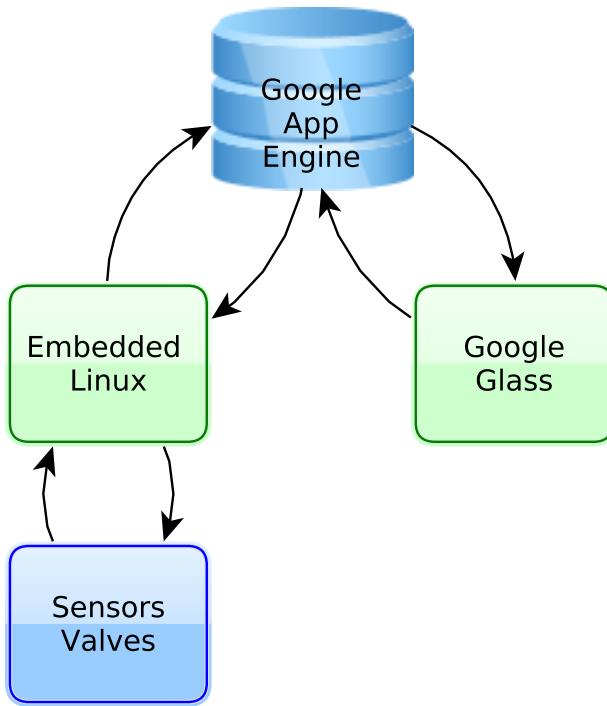


Figure 33: High Level System's Block Diagram

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..

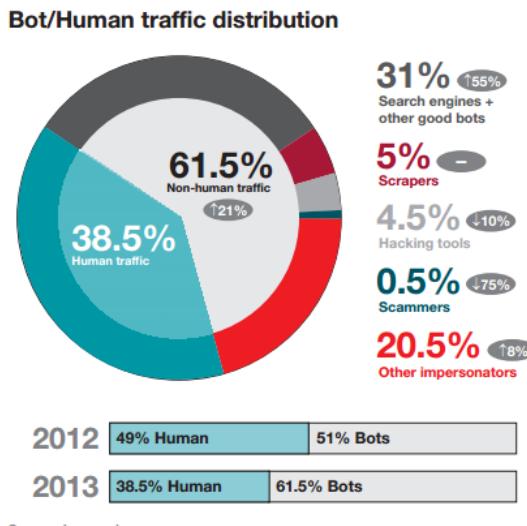


Figure 34: Distribution of Internet of Things

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].

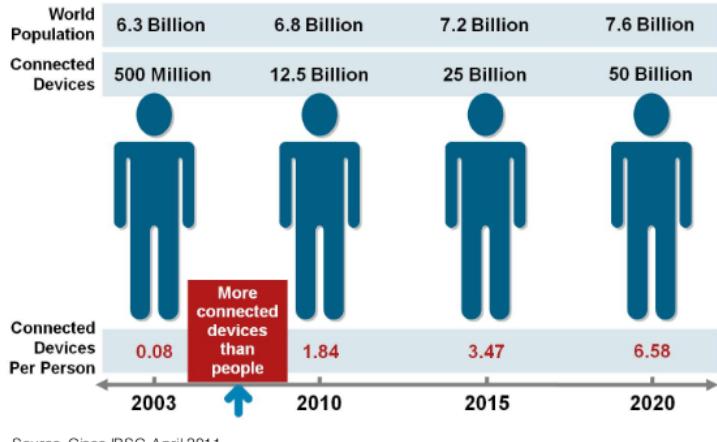


Figure 35: The *IoT* Evolution Prediction

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.

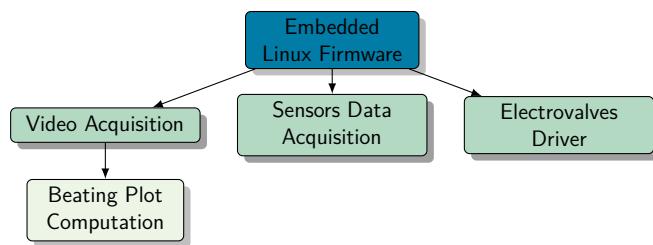


Figure 37: Embedded Linux Firmware - Blocks Subdivision

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.

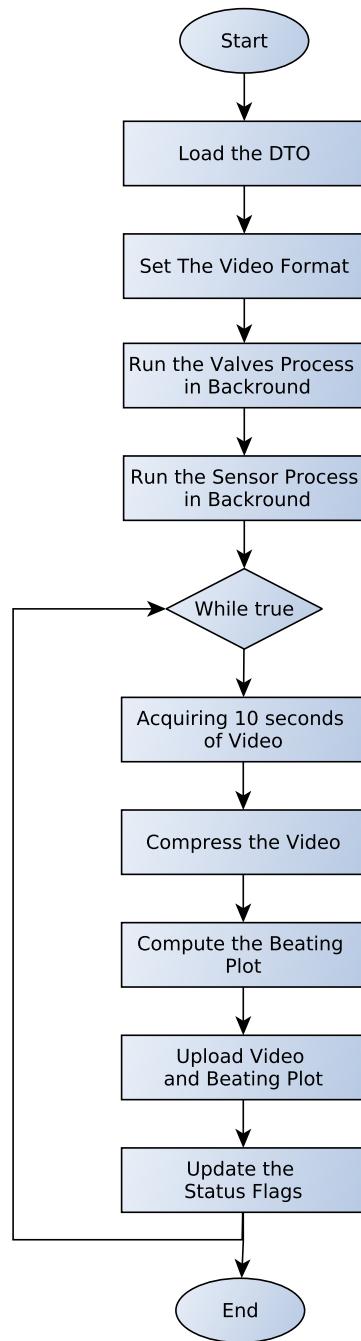


Figure 38: Bash Script Flow Chart

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

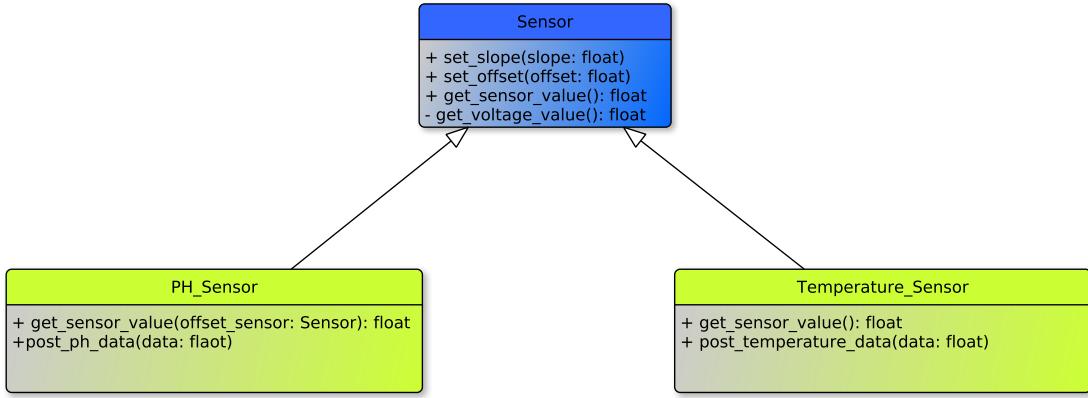
From the main *bash* script another one is launched for acquiring the sensors data. This script is shown in (list.A.2) and as can be seen it is an infinite loop where two operations are performed:

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.4). 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.

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.



Figure 41: Flask Logo

6 | MICROSCOPE VIDEO STORING

Part IV

GOOGLE GLASS APPLICATION

7 | INTRODUCTION

Part V

CONCLUSION AND APPENDIX

8 | TEST AND PERFORMANCE

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

8.1 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.42).

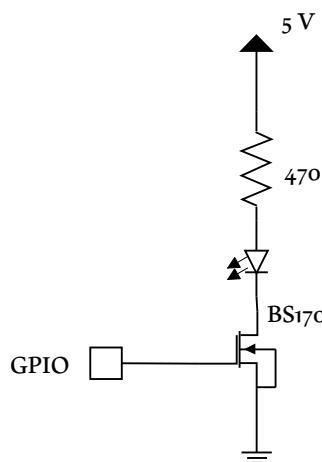


Figure 42: Driver for LED

The (Fig.43) 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.

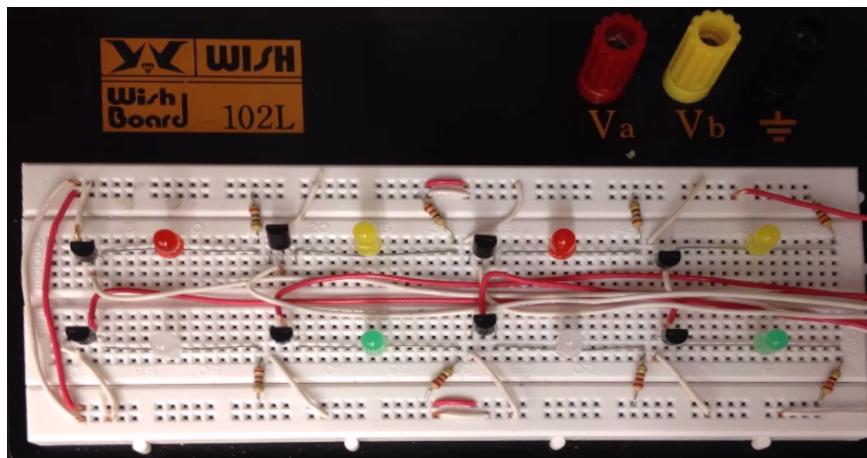


Figure 43: LEDs experiments board

8.2 ELECTROVALVES EXPERIMENTS

8.2.1 Breadboard Phase

The (Fig.44) 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.42), indeed the only difference is given by freewheeling diode, mandatory because of inductive behavior of electrovalve's solenoid.

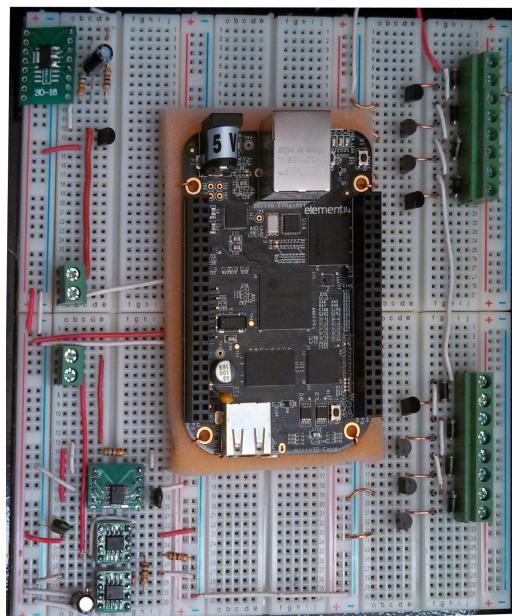


Figure 44: Circuit mounted on breadboard top view

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

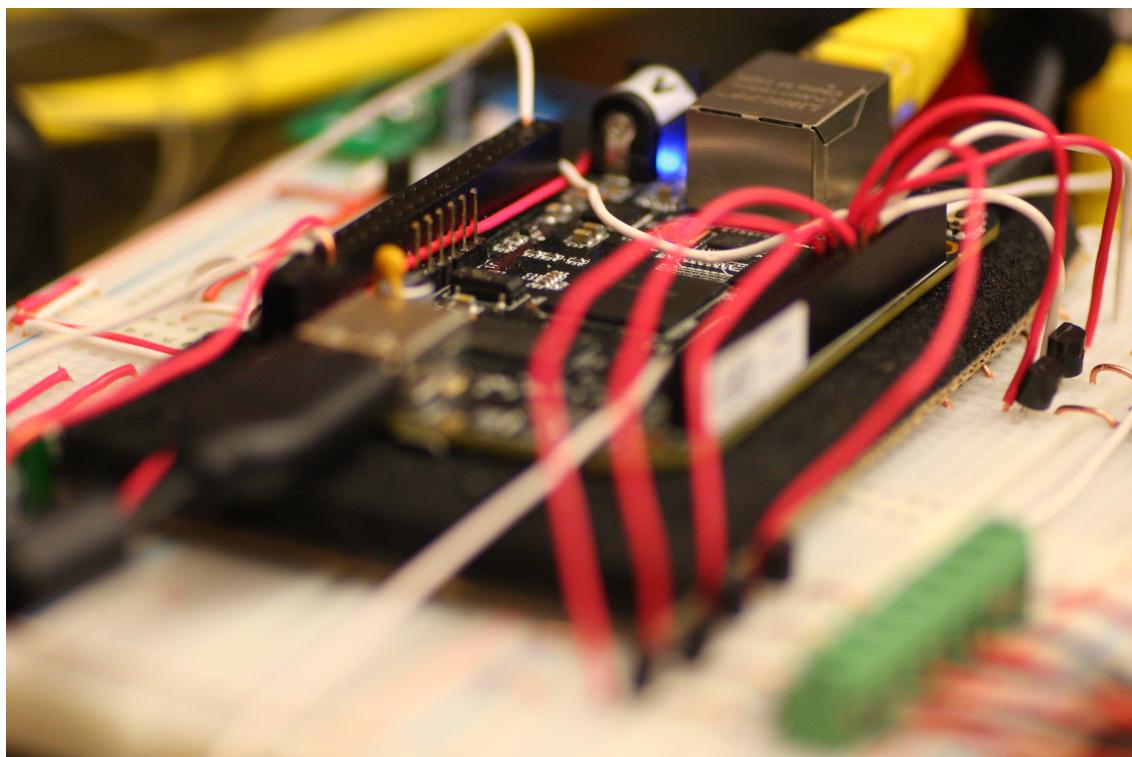


Figure 45: Circuit mounted on breadboard side view

8.2.2 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.

A | CODE

A.1 FIRMWARE

Listing A.1: KlabFirware (bash)

```
1 #!/bin/bash
2 echo "Load the DTO"
3 echo GlassProject-GPIO > $SLOTS
4
5 echo "Setting the video format as MJPG"
6 v4l2-ctl --set-fmt-video=width=320,height=240,pixelformat=1 -d 0
7 v4l2-ctl --set-parm=30
8
9 echo "Running the Electrovalve status updating task"
10 ./Electrovalves/ValvesUpdating </dev/null &>/dev/null &
11 ./Sensors/sensorAcquiring </dev/null &>/dev/null &
12
13 while true; do
14     echo "Recording ten second of Video"
15     ./Video/capture -d /dev/video0 -F -o -c 300 > output.raw -y
16     avconv -f mjpeg -i output.raw -vcodec copy output.mp4 -y
17     echo "Compressing Video"
18     ffmpeg -i output.mp4 -acodec mp2 Video.mp4
19     echo "Uploading the Video to the server"
20     curl -include --form Video.mp4=@Video.mp4 -A "National Instruments LabVIEW"
21         http://planar-contact-601.appspot.com/video/submit -0
22     echo "Updating the flag for video"
23     curl --data "name=video&status=on"
24         http://planar-contact-601.appspot.com/add/electrovalve
25     curl --data "name=glass&status=on"
26         http://planar-contact-601.appspot.com/add/electrovalve
27     echo "Ploting the beating rate"
28     ./Video/compute_beating
29     echo "Uploading the beating plot"
30     curl -include --form Image.jpg=@Image.jpg -A "National Instruments LabVIEW"
31         http://planar-contact-601.appspot.com/picture/submit -0
32     echo "finish"
33 done
```

Listing A.2: SensorAcquiring (bash)

```

1#!/bin/bash

3while true; do
4    echo "Clearing previous data on server"
5    curl http://planar-contact-601.appspot.com/clear
6    echo "Acquiring and uploading new sensors data"
7    ./sensor_acquiring

9done

```

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 = "00A0";
21
22    fragment@0 {
23        target = <&am33xx_pinmux>;
24
25        __overlay__ {
26            ebb_example: KLab {
27                pinctrl-single,pins = <
28
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        };
42    };
43}

```

```

37          0x0a0 0x07 // P8_46 - PowerSupply

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

43  fragment@1 {
44  target = <&ocp>;
45  __overlay__ {
46  gpiod_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://planar-contact-601.appspot.com/show/";
14 std::string const TRUE_VALUE = " True ";
15 std::string const FALSE_VALUE = " False ";

16
17 static size_t WriteCallback(void *contents, size_t size, size_t nmemb, void *userp)
18 {
19   ((std::string*)userp)->append((char*)contents, size * nmemb);
20   return size * nmemb;
21 }

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

```

```

29     CURL *curl;
30     CURLcode res;
31
32     GPIO_VALUE output_vector[NUMBER_ELECTROVALVES];
33
34     //Set output pins
35     EV1.setDirection(OUTPUT);
36     EV2.setDirection(OUTPUT);
37     EV3.setDirection(OUTPUT);
38     EV4.setDirection(OUTPUT);
39     EV5.setDirection(OUTPUT);
40     EV6.setDirection(OUTPUT);
41     EV7.setDirection(OUTPUT);
42     EV8.setDirection(OUTPUT);
43
44     RELAY.setDirection(OUTPUT);
45
46     RELAY.setValue(HIGH);
47
48     while(1)
49     {
50         //acquire the values of electrovalves status and store them inside
51         //output_vector
52         int i;
53         for(i=0;i<NUMBER_ELECTROVALVES; i++)
54         {
55             curl = curl_easy_init();
56             if(curl)
57             {
58                 std::string readBuffer;
59                 std::ostringstream ss;
60                 ss << i+1;
61                 std::string url = ELECTROVALVES_LINK + "EV" + ss.str();
62                 curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
63                 /* example.com is redirected, so we tell libcurl to follow redirection */
64                 curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
65                 curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
66                 /* Perform the request, res will get the return code */
67                 res = curl_easy_perform(curl);
68                 /* always cleanup */
69                 curl_easy_cleanup(curl);
70
71                 if(readBuffer.compare(TRUE_VALUE) == 0)
72                 {
73                     std::cout << "EV" + ss.str() + ": ON" << std::endl;
74                     output_vector[i] = HIGH;
75                 }
76                 else if(readBuffer.compare(FALSE_VALUE) == 0)
77                 {
78                     std::cout << "EV" + ss.str() + ": OFF" << std::endl;
79                     output_vector[i] = LOW;
80                 }
81             }
82         }
83     }
84 }
```

```

77     {
78         std::cout << "EV" + ss.str() + ": OFF" << std::endl;
79         output_vector[i] = LOW;
80     }
81 }
82
83 //now I'm ready to write the output pins:
84 EV1.setValue(output_vector[0]);
85 EV2.setValue(output_vector[1]);
86 EV3.setValue(output_vector[2]);
87 EV4.setValue(output_vector[3]);
88 EV5.setValue(output_vector[4]);
89 EV6.setValue(output_vector[5]);
90 EV7.setValue(output_vector[6]);
91 EV8.setValue(output_vector[7]);
92 }
93 return 0;
94 }
```

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"
10 using namespace std;
11
12 #define LDR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"
13 #define GPIO_PATH "/sys/class/gpio/"
14 #define OUTPUT false
15 #define INPUT true
16 #define HIGH true
17 #define LOW false
18
19 float const CURRENT = 0.00068;
20 float const SLOPE_TEMPERATURE = 2;
21 float const OFFSET_TEMPERATURE = 1870;
22 float const SLOPE_PH = -0.070;
23 float const OFFSET_PH = 0.6;
24
25 class Sensor
26 {
27     int number;
28 }
```

```

public:
30   float m;
31   float q;
32 Sensor(int x): number(x){}
33   void set_slope(float slope){
34     m = slope;
35   }
36   void set_offset(float offset){
37     q = offset;
38   }
39   float get_sensor_value(){
40     float voltage_value = this->get_voltage_value();
41     float sensor_value = (voltage_value - q)/m;
42     return sensor_value;
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 };
58

60 class Temperature_Sensor : public Sensor{
61 private:
62
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{

```

```
78 public:
79     PH_Sensor(int x) : Sensor(x) {}
80     float get_sensor_value(Sensor offset_sensor){
81         //Sensor_offset_sensor(2);
82         float voltage_value = this->get_voltage_value();
83         q += offset_sensor.get_voltage_value();
84         float sensor_value = (voltage_value - q)/m;
85         return sensor_value;
86     }
87
88     void post_ph_data(float data){
89         post_data_sensor(1, data);
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     }
108     else
109     {
110         slope_temperature = SLOPE_TEMPERATURE;
111         offset_temperature = OFFSET_TEMPERATURE;
112         slope_ph = SLOPE_PH;
113         offset_ph = OFFSET_PH;
114     }
115
116     //set the pin GPIO_45 as output
117     temperature_disable.setDirection(OUTPUT);
118     //Temporary disable the temperature
119     temperature_disable.setValue(HIGH);
120
121     Temperature_Sensor temperature_sensor(2);
122     PH_Sensor ph_sensor(1);
123     Sensor offset_sensor(2);
124
125     temperature_sensor.set_slope(slope_temperature);
126     temperature_sensor.set_offset(offset_temperature);
```

```

    ph_sensor.set_slope(slope_ph);
128  ph_sensor.set_offset(offset_ph);

130  int i= 0;
131  for(i=0; i<100; i++)
132  {
133      temperature_disable.setValue(LOW);
134      float ph = ph_sensor.get_sensor_value(offset_sensor);
135      float temperature = temperature_sensor.get_sensor_value();
136      temperature_disable.setValue(HIGH);
137      cout << "The voltage value is: " << temperature << " V." << endl;
138      temperature_sensor.post_temperature_data(temperature);
139      ph_sensor.post_ph_data(ph);
140      usleep(100000);
141  }
142  float temperature = temperature_sensor.get_voltage_value();
143  cout << "The voltage value is: " << temperature << " V." << endl;
144  return 0;
}

```

Listing A.6: compute_beating.cpp

```

1
2 #include<iostream>
3 #include<fstream>
4 #include<string>
5 #include <curl/curl.h>
6 #include <sys/stat.h>
7 #include <fcntl.h>
8 #include<sstream>
9 #include<opencv2/opencv.hpp> // C++ OpenCV include file
10 using namespace std;
11 using namespace cv; // using the cv namespace too

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");

21     // set any properties in the VideoCapture object
22     capture.set(CV_CAP_PROP_FRAME_WIDTH,320); // width pixels
23     capture.set(CV_CAP_PROP_FRAME_HEIGHT,240); // height pixels

25     if(!capture.isOpened())
26     {
27         cout << "Capture not open." << endl;
28     }

```

```

29     Mat frame, firstFrame, diffFrame;

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;

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         float mean_color = (red+green+blue)/3;

53         //qui poi chiamo la funzione che mi posta il punto nel db
54         float temp = (i-1);
55         temp *= 0.033;

57         point_file << temp << '\t';
58         point_file << mean_color << '\n';

59     }

61 }

63
64     point_file.close();
65     return 0;
66 }
```

Listing A.7: Pins Setting

```

/*
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/;
16
17 //{
18     compatible = "ti,beaglebone", "ti,beaglebone-black";
19     part-number = "KLab";
20     version = "00AO";
21
22     fragment@0 {
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   };
43
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
4 #include<iostream>
5 #include<fstream>
6
7 #include<unistd.h>
8
9 #include<string>
10
11 #include<sstream>
12
13 #include<stdlib.h>
14
15 #include<curl/curl.h>
16
17 void post_data_sensor(int type, float value);
18
19#endif // HTTP
```

Listing A.9: HTTP.cpp

```
#include "http.h"

void post_data_sensor(int type, float value){
    CURL *curl;
    CURLcode res;
    /* In windows, this will init the winsock stuff */
    curl_global_init(CURL_GLOBAL_ALL);

    /* get a curl handle */
    curl = curl_easy_init();
    if(curl) {
        /* First set the URL that is about to receive our POST. This URL can
           just as well be a https:// URL if that is what should receive the
           data. */
        curl_easy_setopt(curl, CURLOPT_URL, URL_POST.c_str());
        /* Now specify the POST data */
        std::ostringstream ss;
        ss << value;
        std::string s;
        if (type == 0)
            s= POST_DATA_TEMPERATURE +(ss.str());
        else if (type == 1)
            s= POST_DATA_PH +(ss.str());
        else
            s= POST_DATA_BEATING +(ss.str());
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, s.c_str());
    }
}
```

```

28     /* Perform the request, res will get the return code */
29     res = curl_easy_perform(curl);
30     /* Check for errors */
31     if(res != CURLE_OK)
32         fprintf(stderr, "curl_easy_perform() failed: %s\n",
33                 curl_easy_strerror(res));
34     /* always cleanup */
35     curl_easy_cleanup(curl);
36 }
37 curl_global_cleanup();
38 }
```

A.2 VIDEO STORING SOFTWARE

Listing A.10: main.cpp

```

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

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
16 public slots:
17     void timerSlot();
18 private:
19     Downloader d;
20 };
21
22 #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()));

```

```

9     timer->start(10000);

11 }

13 MyTimer::~MyTimer()
{
15 }
17

19

21 void MyTimer::timerSlot(){
    d.checkFlag();
23    if( d.getStatus() )
    {
25        d.resetFlag();
        d.doDownload();
27    }
29}

```

Listing A.14: downloader.h

```

1 #ifndef DOWNLOADER_H
2 #define DOWNLOADER_H

4 #include <QObject>
5 #include <QNetworkAccessManager>
6 #include <QNetworkRequest>
7 #include <QNetworkReply>
8 #include <QUrl>
9 #include <QDateTime>
10 #include <QFile>
11 #include <QDebug>
12 #include <QEventLoop>

14 class Downloader : public QObject
{
16     Q_OBJECT
17
18     public:
19         explicit Downloader(QObject *parent = 0);
20
21         void doDownload();
22         void checkFlag();
23         bool getStatus();
24         void resetFlag();

```

```

signals:
26
public slots:
28     void replyFinished (QNetworkReply *reply);
29     void checkStatus(QNetworkReply *replay);
30
private:
32     QNetworkAccessManager *manager;
33     QNetworkAccessManager *manager2;
34     QDateTime data;
35     QString name;
36     bool flag_ready;
37     bool status;
38
39 };
40
41 #endif // DOWNLOADER_H

```

Listing A.15: downloader.cpp

```

#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
15     manager->get(QNetworkRequest(QUrl("http://planar-contact-601.appspot.com/video/view")));
16 }
17
18 void Downloader::replyFinished (QNetworkReply *reply)
19 {
20     if(reply->error())
21     {
22         qDebug() << "ERROR!";
23         qDebug() << reply->errorString();
24     }
25     else
26     {
27         qDebug() << reply->header(QNetworkRequest::ContentTypeHeader).toString();
28     }
29 }

```

```

    qDebug() << reply->header(QNetworkRequest::LastModifiedHeader).toDateTime().toString();
30   qDebug() << reply->header(QNetworkRequest::ContentLengthHeader).toULongLong();
    qDebug() << reply->attribute(QNetworkRequest::HttpStatusCodeAttribute).toInt();
32   qDebug() << reply->attribute(QNetworkRequest::HttpReasonPhraseAttribute).toString();

34   data = QDateTime::currentDateTime();
35   name = "C:/Video/" + data.toString("yyyy-MM-dd-HH-mmss") + ".mpeg";
36
37   QFile *file = new QFile(name);
38   if(file->open(QFile::Append))
39   {
40       file->write(reply->readAll());
41       file->flush();
42       file->close();
43   }
44   delete file;
45 }

46   reply->deleteLater();
47 }

50
51 void Downloader::checkStatus(QNetworkReply *replay)
52 {
53     QString result(replay->readAll());
54     int compare = QString::compare(result, FLAG_ON);
55     if(compare == 0)
56     {
57         status = true;
58     }
59     else
60     {
61         status = false;
62     }
63     qDebug() << status;
64     flag_ready = true;
65 }

66 }
67 void Downloader::checkFlag()
68 {
69     manager = new QNetworkAccessManager(this);
70     QNetworkReply *reply = manager->get(QNetworkRequest(QUrl("http://planar-contact-601.appspot.com/")));
71     QEeventLoop loop;
72     connect(reply, SIGNAL(finished()), &loop, SLOT(quit()));

73
74     loop.exec();
75     qDebug() << "I'm looking for a new video";
76     QString result(reply->readAll());
77     int compare = QString::compare(result, FLAG_ON);

```

```

78     if(compare == 0)
79     {
80         status = true;
81         qDebug() << "Found it";
82     }
83     else
84     {
85         status = false;
86         qDebug() << "No new video available";
87     }
88 }
89
90     bool Downloader::getStatus()
91 {
92     return status;
93 }
94
95 void Downloader::resetFlag()
96 {
97     status = false;
98     manager2 = new QNetworkAccessManager(this);
99
100    QUrl flag_url = QUrl("http://planar-contact-601.appspot.com/add/electrovalve");
101    QByteArray postData;
102    postData.append("name=video&status=off");
103    manager2->post(QNetworkRequest(flag_url), postData);
104 }

```

A.3 GOOGLE APP ENGINE

Listing A.16: Main Script of Server

```

1 # the application id inside app.yaml has to match with the ID in google app engine
2
3 from flask import Flask
4
5 app = Flask(__name__)
6
7 from flask import request
8 from flask import make_response
9
10 from models import MESSAGES
11 from google.appengine.ext import ndb
12 from google.appengine.api import memcache
13 import logging

```

```
import datetime
15 import cloudstorage
16 import json
17
key_list = []
19
NUMBER_OF_ELECTROVALVES = 8
21 SECONDS_BETWEEN_UPDATES = 60
22 SENSOR_TIMEOUT_IN_SECONDS = 300
23 ELECTROVALVES_TIME_OUT = 1000
24 BUCKET_NAME = "/planar-contact-601.appspot.com/"
25 ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'mp4'}
26 UPDATE_FLAG = 'database_updated_recently'
27
28
29 class DataPoint(ndb.Model):
    value = ndb.FloatProperty()
30     date = ndb.DateTimeProperty(auto_now_add=True)
31
32     @classmethod
33     def points_for_sensor(cls, sensor_key):
34         return cls.query(ancestor=sensor_key).order(-cls.date)
35
36     @classmethod
37     def oldest_points(cls):
38         return cls.query().order(+cls.date)
39
40
41 class Sensor(ndb.Model):
42     @classmethod
43     def sensor_list(cls):
44         return cls.query()
45
46
47 def update():
48     # Check to see if we have updated recently
49     update_flag = memcache.get(UPDATE_FLAG)
50     if update_flag is not None:
51         return
52
53     memcache.add(UPDATE_FLAG, 'YES', SECONDS_BETWEEN_UPDATES)
54
55     # Delete old data points
56     now = datetime.datetime.now()
57     count = 0
58
59     for point in DataPoint.oldest_points().iter():
60         delta = (now - point.date).total_seconds()
61         if delta > 3600:
62             point.key.delete()
63             count += 1
64         else:
```

```

63         break
64     logging.info("Deleted {} old data points".format(count))
65     # Loop through all of the sensors
66     sensor_names = []
67     for sensor in Sensor.sensor_list().iter():
68         # Store current values of each sensor
69         sensor_value = memcache.get(sensor.key.id() + "_value")
70         if sensor_value is not None:
71             point = DataPoint(parent=sensor.key, value=sensor_value)
72             point.put()
73         # Delete sensors that don't have any data
74         curr_points = DataPoint.points_for_sensor(sensor.key)
75         if curr_points.count() == 0 and sensor_value is None:
76             logging.info("Deleting sensor {}".format(sensor.key.id()))
77             sensor.key.delete()
78         else:
79             sensor_names.append(sensor.key.id())
80     # Store a list of sensor names in memcache
81     memcache.set("sensor_names", ",".join(sensor_names))

83
84     @app.route('/', methods=['POST'])
85     @app.route('/sensor_values', methods=['GET', 'POST'])
86     def process_values():
87         if request.method == 'GET':
88             sensor_names = memcache.get('sensor_names')
89             if sensor_names is None:
90                 sensor_names = ",".join([sensor.key.id() for sensor in Sensor.sensor_list()])
91                 memcache.set('sensor_names', sensor_names)
92             sensor_names = sensor_names.split(",")
93             sensor_values = {}
94             for sensor_name in sensor_names:
95                 sensor_value = memcache.get(sensor_name + '_value')
96                 if sensor_value is not None:
97                     sensor_values[sensor_name] = sensor_value
98             return json.dumps(sensor_values)
99         elif request.method == 'POST':
100             update()
101             logging.info(request.values)
102             sensor_name = request.form.get('sensor')
103             sensor_value = float(request.form.get('value'))
104             sensor_names = memcache.get('sensor_names')
105             if sensor_names is not None and sensor_name not in sensor_names:
106                 sensor = Sensor()
107                 sensor.key = ndb.Key('Sensor', sensor_name)
108                 sensor.put()
109                 sensor_names = sensor_name if sensor_names == "" else sensor_names + "," + sensor_name
110                 memcache.set('sensor_names', sensor_names)
111                 memcache.set(sensor_name + '_value', sensor_value, SENSOR_TIMEOUT_IN_SECONDS)

```

```

    return "Success\n"
113

115 @app.route('/sensor_names', methods=['GET'])
116 def print_names():
117     return json.dumps([sensor.key.id() for sensor in Sensor.sensor_list()])
118

119 @app.route('/graphing_data', methods=['GET'])
120 def graphing_data():
121     last_timestamp = request.args.get('last_timestamp')
122     sensor_name = request.args.get('sensor')
123     if last_timestamp is None:
124         last_timestamp = 0
125     else:
126         last_timestamp = float(last_timestamp)
127     points = DataPoint.points_for_sensor(ndb.Key('Sensor', sensor_name))
128     epoch = datetime.datetime(1970, 1, 1)
129     res = []
130     for point in points.iter():
131         timestamp = (point.date - epoch).total_seconds()
132         if timestamp > last_timestamp:
133             _res = {"timestamp": timestamp, "value": point.value}
134             res.append(_res)
135         else:
136             break
137     res.reverse()
138     return json.dumps(res)

141
142 @app.route('/clear', methods=['GET'])
143 def clear_data():
144     points = DataPoint.query()
145     for point in points:
146         point.key.delete()
147     for sensor in Sensor.sensor_list():
148         sensor.key.delete()
149     return "Success"

151
152 @app.route('/picture/submit', methods=['GET', 'POST'])
153 def set_picture():
154     if request.method == 'POST':
155
156         _file = request.files['Image.jpg']
157
158         if _file:
159             cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_image." + _file.filename.rsplit(

```

```
161         _file.save(cloud_file)
162         cloud_file.close()
163     return "Success"
164
165     else:
166         return ''
167
168         <!doctype html>
169         <title>Upload microscope file V.1</title>
170         <h1>Upload microscope file</h1>
171         <form method="POST"
172             action=""
173             role="form"
174             enctype="multipart/form-data">
175
176             <p><input type=file name=Image.jpg>
177                 <input type=submit value=Upload>
178             </form>
179             ''
180
181
182 @app.route('/picture/view', methods=['GET'])
183 def view_picture():
184     for _file in cloudstorage.listbucket(BUCKET_NAME):
185         if "microscope_image" in _file.filename:
186             _file = cloudstorage.stat(_file.filename)
187             logging.info(_file.filename)
188             logging.info(_file.content_type)
189             cloud_file = cloudstorage.open(_file.filename, mode='r')
190             response = make_response(cloud_file.read())
191             cloud_file.close()
192             response.mimetype = _file.content_type
193             return response
194
195     return "No file found"
196
197
198 @app.route('/video/submit', methods=['GET', 'POST', 'PUT'])
199 def set_video():
200     if request.method == 'GET':
201         return ''
202
203         <!doctype html>
204         <title>Upload microscope video</title>
205         <h1>Upload microscope video</h1>
206         <form method="POST"
207             action=""
208             role="form"
209             enctype="multipart/form-data">
210             <p><input type=file name=Video.mp4>
211                 <input type=submit value=Upload>
212             </form>
213             ''
```

```

    else:
        _file = request.files['Video.mp4']
        if _file:
            cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_video." + _file.filename.rsplit(
                mode='w', content_type="video/mpeg")
            _file.save(cloud_file)
            cloud_file.close()
    return "Success"

219

221 @app.route('/video/view', methods=['GET'])
def view_video():
    for _file in cloudstorage.listbucket(BUCKET_NAME):
        if "microscope_video" in _file.filename:
            _file = cloudstorage.stat(_file.filename)
            logging.info(_file.filename)
            logging.info(_file.content_type)
            cloud_file = cloudstorage.open(_file.filename, mode='r')
            response = make_response(cloud_file.read())
            cloud_file.close()
            response.mimetype = _file.content_type
            return response
    return "No file found"

235

236 @app.errorhandler(404)
def page_not_found(e):
    """Return a custom 404 error."""
    return 'Sorry, nothing at this URL.', 404

241

242 class Electrovalve(ndb.Model):
    # name = ndb.StringProperty()
    # status = ndb.BooleanProperty() # if the status is true, the Electrovalve is on
    # date = ndb.DateTimeProperty(auto_now_add=True)
    #
247    # def __init__(self, name, status, **kwds):
    #     super(Electrovalve, self).__init__(**kwds)
    #     self.name = name
    #     self.status = status
    #
251    # @classmethod
252    # def get_status(cls):
    #     return cls.status
    #
255    # @classmethod
256    # def get_ev_by_name(cls, ev_name):
    #     return cls.query(ancestor=ev_name).fetch

```

```

259
260     @classmethod
261     def ev_list(cls):
262         return cls.query()
263
264
265 # @app.route('/add/<key>/<message>')
266 # def update_message(key, message):
267 #     if key and message:
268 #
269 #         if message == 'on' or message == 'On' or message == 'ON':
270 #             MESSAGES[key] = True
271 #
272 #         else:
273 #             MESSAGES[key] = False
274 #
275 #         # temp = Electrovalve(name=key, status=message)
276 #         # temp.put()
277 #     return "%s Updated" % key
278
279
280 @app.route('/electrovalves/<name>', methods=['GET'])
281 def electrovalve(name):
282     ev = memcache.get(name);
283     if ev is None:
284         return '%r' % MESSAGES[name] or "%s not found!" % name
285     else:
286         return '%r' % memcache.get(name)
287     #     ev_values = {}
288     #     # return "boia"
289     #     #     ev_names = ",".join([ev.key.id() for ev in Electrovalve.ev_list()])
290     #     #     memcache.set('ev_names', ev_names)
291     #     #     ev_names = ev_names.split(",")
292     #     #     ev_values = {}
293     #     #     for ev_name in ev_names:
294     #     #         ev_value = memcache.get(ev_name + '_value')
295     #
296     #         if ev_value is not None:
297     #             ev_values[ev_name] = ev_value
298     #
299     #     return '%r' % ev_values[name]
300
301     #
302     # # ev_name =
303     # ev_key = ndb.Key("EV", name or "*notitle")
304     # ev = Electrovalve.query(name)
305     # return '%r' % ev.st
306
307

```

```

309 @app.route('/add/electrovalve', methods=['POST'])
310 def add_electrovalve():
311     key = request.form.get('name')
312     message = request.form.get('status')
313     if key and message:
314         if message == 'on' or message == 'On' or message == 'ON':
315             MESSAGES[key] = True
316             memcache.set(key, True)
317
318         else:
319             MESSAGES[key] = False
320             memcache.set(key, False)
321
322         # ev.put()
323     return '%r' % memcache.get(key)
324
325 @app.route('/glass')
326 def glass():
327     return "Buso e' qui"
328
329 @app.route("/show/<key>")
330 def get_message(key):
331     return '%r' % MESSAGES[key] or "%s not found!" % key
332
333
334 if __name__ == "__main__":
335     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
41 import java.io.BufferedInputStream;
42 import java.io.File;
43 import java.io.FileOutputStream;
44 import java.io.IOException;
45 import java.io.InputStream;
46 import java.net.HttpURLConnection;
47 import java.net.URL;
48 import java.net.URLConnection;
49 import java.nio.channels.FileLock;
50 import java.util.ArrayList;
51 import java.util.Collections;
52 import java.util.HashMap;
53 import java.util.Map;
54
55 /**
56  * Service owning the LiveCard living in the timeline.
57  */
58
59 public class MainService extends Service {
60     /** TAG associated to the LiveCard */
61     private static final String LIVE_CARD_TAG = "BWH interface";
```

```
63     /** TAG associated to the Menu view */
64     private static final String MENU_TAG = "Menu";
65     /** TAG associated to the Temperature view */
66     private static final String TEMPERATURE_TAG = "Temperature";
67     /** TAG associated to the pH view */
68     private static final String pH_TAG = "pH";
69     /** TAG associated to the Video view */
70     private static final String VIDEO_TAG = "Video";
71     /** TAG associated to the Beating view */
72     private static final String BEATING_TAG = "Beating";

73     private Bitmap bmp;

74     /** URL where the image is stored */
75     private static final String URL_IMAGE = "http://planar-contact-601.appspot.com/picture/view";
76

77

78

79     /** Runnable which describes the task to download the Beating's graph */
80     private final ImageDownloader mImageDownloader = new ImageDownloader(URL_IMAGE, this);
81     /** Action is an enumerate used to implement switch-case for the extra text appended to the intent
82     private static enum Action
83     {
84         Menu, Temperature, pH, Video, Beating
85     }
86

87

88     private AppDrawer mCallback;

89

90     private LiveCard mLiveCard;

91

92     /** HandlerThread used to launch a background thread that manages the Data updating */
93     private HandlerThread mHandlerThread;
94     /** Handler used to launch a background thread that manages the Data updating */
95     private Handler mHandler;
96

97     /** INT associated to the Menu view request */
98     private static final int MENU = 0;
99     /** INT associated to the pH view request */
100    private static final int pH = 1;
101    /** INT associated to the Menu view request */
102    private static final int TEMPERATURE = 2;
103    /** INT associated to the Video view request */
104    private static final int VIDEO = 3;
105    /** INT associated to the Beating view request */
106    private static final int BEATING = 4;

107

108    /** updating data period (in ms) */
109    private static final long DATA_UPDATE_DELAY_MILLIS = 500;
```

```

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

117  /** Runnable which describes the task to update the pH and Temperature sensors values  */
118  private final UpdateSensorValuesRunnable mUpdateSensorValuesRunnable = new UpdateSensorValuesRunn
119  /** Runnable which describes the task to compute the pH and Temperature graph (starting from the
120  private final UpdateSensorGraphsRunnable mUpdateSensorGraphsRunnable = new UpdateSensorGraphsRunn
121
122  private final UpdateMicroscopeVideoRunnable mUpdateMicroscopeVideoRunnable = new UpdateMicroscop
123
124  private static final String VIDEO_FILE_NAME = Environment.getExternalStorageDirectory() + "/microsc
125  private static final String TEMP_VIDEO_FILE_NAME = Environment.getExternalStorageDirectory() + "/temp

127  /** String array for the sensors (PH and Temperature) */
128  private String[] mSensors = new String[]{PH_TAG, TEMPERATURE_TAG};

129
130  /*
131   *          Hash table used for creating the graphs
132  */
133  /** Hash table in which the sensors values are stored */
134  private Map<String, Double> mCurrentSensorValues;
135  /** Hash table in which the graphs points are stored */
136  private Map<String, ArrayList<DataPoint>> mSensorGraphData;
137  /** Hash table in which the sensors graphs are stored */
138  private Map<String, Bitmap> mCurrentSensorGraphs;

139
140  //  private ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_S
141

143  @Override
144  public IBinder onBind(Intent intent) {
145      return null;
146  }

147  @Override
148  public int onStartCommand(Intent intent, int flags, int startId) {
149      if (mLiveCard == null) {
150          AppManager.getInstance().setState(MENU);
151          mLiveCard = new LiveCard(this, LIVE_CARD_TAG);

153
154      // Keep track of the callback to remove it before unpublishing.
155      mCallback = new AppDrawer(this);
156      mLiveCard.setDirectRenderingEnabled(true).getSurfaceHolder().addCallback(mCallback);

157
158      Intent menuIntent = new Intent(this, MenuActivity.class);
159      menuIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);

```

```

161     mLIVECARD.setAction(PendingIntent.getActivity(this, 0, menuIntent, 0));
162     mLIVECARD.attach(this);
163     mLIVECARD.publish(PublishMode.REVEAL);

164
165     /*           Launch the task to update the data in another thread
166     mHandlerThread = new HandlerThread("myHandlerThread");
167     mHandlerThread.start();
168     mHandler = new Handler(mHandlerThread.getLooper());
169     DataTask task = new DataTask();
170     task.execute(this);
171 } else {
172     if(intent.getStringExtra(Intent.EXTRA_TEXT) != null) {
173         Action action = Action.valueOf(intent.getStringExtra(Intent.EXTRA_TEXT));
174         switch (action) {
175             case Menu:
176                 Log.i(LIVE_CARD_TAG, "State = Menu");
177                 AppManager.getInstance().setState(MENU);
178                 break;
179             case Beating:
180                 Log.i(LIVE_CARD_TAG, "State = Beating");
181                 AppManager.getInstance().setState(BEATING);
182                 break;
183             case pH:
184                 Log.i(LIVE_CARD_TAG, "State = pH");
185                 AppManager.getInstance().setState(PH);
186                 break;
187             case Temperature:
188                 Log.i(LIVE_CARD_TAG, "State = Temperature");
189                 AppManager.getInstance().setState(TEMPERATURE);
190                 break;
191             case Video:
192                 Log.i(LIVE_CARD_TAG, "State = Video");
193                 AppManager.getInstance().setState(VIDEO);
194                 break;
195             default:
196                 mLIVECARD.navigate();
197                 break;
198             }
199         }
200     else
201     {
202         mLIVECARD.navigate();
203     }
204 }

// Return START_NOT_STICKY to prevent the system from restarting the service if it is killed
// (e.g., due to an error). It doesn't make sense to restart automatically because the
// stopwatch state will have been lost.

```

```

209         return START_NOT_STICKY;
210     }
211
212     @Override
213     public void onDestroy() {
214         // Stop the Task which update the beating image
215         if (!mImageDownloader.isStopped())
216         {
217             mImageDownloader.setIsStopped(true);
218             Log.i(BEATING_TAG, "Removed Task");
219         }
220         // Stop the Task which update the sensors Graphs
221         if (!mUpdateSensorGraphsRunnable.isStopped())
222         {
223             mUpdateSensorGraphsRunnable.setStop(true);
224             Log.i(PH_TAG + " " + TEMPERATURE_TAG, "Removed Task of Graphs");
225         }
226         // Stop the Task which update the sensors value
227         if (!mUpdateSensorValuesRunnable.isStopped())
228         {
229             mUpdateSensorValuesRunnable.setStop(true);
230             Log.i(PH_TAG + " " + TEMPERATURE_TAG, "Removed Task of Values");
231         }
232         if (!mUpdateMicroscopeVideoRunnable.isStopped())
233         {
234             mUpdateMicroscopeVideoRunnable.setStop(true);
235             Log.i(VIDEO_TAG, "Removed Task of Uploading values");
236         }
237         if (mLiveCard != null && mLiveCard.isPublished()) {
238             mLiveCard.unpublish();
239             mLiveCard = null;
240         }
241         super.onDestroy();
242     }
243
244     /**
245      * Asynchronous Task for downloading the graphs and video in background
246      */
247     private class DataTask extends AsyncTask<MainService,Void(Void>
248     {
249
250         @Override
251         protected Void doInBackground(MainService... params) {
252
253             Log.i(LIVE_CARD_TAG, "Loading initial data");
254
255             // create the three hash tables
256             mCurrentSensorValues = new HashMap<String, Double>();
257             mSensorGraphData = new HashMap<String, ArrayList<DataPoint>>();

```

```

259         mCurrentSensorGraphs = new HashMap<String, Bitmap>();
260         mSensorAverage = new HashMap<String, Double>();

261         // initializes each hash map with dummy values
262         for (String mSensor : mSensors)
263         {
264             mCurrentSensorValues.put(mSensor, 0.0);
265             mSensorAverage.put(mSensor, 0.0);
266             mSensorGraphData.put(mSensor, new ArrayList<DataPoint>());
267             mCurrentSensorGraphs.put(mSensor, null);
268         }

269         Log.i(LIVE_CARD_TAG, "Download the data");
270
271         //uploads the value captured by PH and Temperature sensors
272         //NetworkInfo ni = cm.getActiveNetworkInfo();
273         //if(ni!= null) {
274
275             // if (ni.isConnected()) {
276                 mUpdateSensorValuesRunnable.run();
277                 // give the hash table to the Callback
278                 // mCallback.setSensorValues(mCurrentSensorValues);
279                 // compute the graphs with previous values and give them to the AppDrawer
280                 mUpdateSensorGraphsRunnable.run();
281
282                 // give the hash table to the Callback
283                 // mCallback.setSensorGraphs(mCurrentSensorGraphs);
284                 // download the beating image and give this to the AppDrawer
285                 mImageDownloader.run();
286
287                 mUpdateMicroscopeVideoRunnable.run();
288             //}
289         //}
290
291
292
293
294         return null;
295     }
296
297
298 /**
299 * This runnable updates the sensors values taking them from the google engine
300 */
301 private class UpdateSensorValuesRunnable implements Runnable
302 {
303     private boolean mIsStopped = false;
304
305     /** It implements the task of runnable

```

```
307     *
308     * @see UpdateSensorValuesRunnable
309     */
310
311     @Override
312     public void run()
313     {
314
315         if (!isStopped())
316         {
317
318             /** JavaScript object in which the sensor values are temporary stored */
319             JSONObject values = getSensorValues();
320
321             if (values != null)
322             {
323
324                 for (String mSensor : mSensors)
325                 {
326
327                     try
328                     {
329
330                         // update the hash table of sensor values
331                         mCurrentSensorValues.put(mSensor, values.getDouble(mSensor));
332
333                     }
334
335                     catch (JSONException ignored)
336                     {}
337
338                 }
339             }
340
341             // restart the Runnable after a given amount of time
342             mHandler.postDelayed(mUpdateSensorValuesRunnable, DATA_UPDATE_DELAY_MILLIS);
343
344         }
345     }
346
347
348
349
350
351
352
353
354
355
```

```
        }

357     /** It gets the values of all of the sensors
359      *
360      * @return JSONObject which contains all the values of sensors
361      */
362     private JSONObject getSensorValues()
363     {
364         try
365         {
366             String values = getURL("http://planar-contact-601.appspot.com/sensor_values");
367             // return a JSONObject constructed by JSONTokener, which takes a source string and extracts
368             return new JSONObject(new JSONTokener(values));
369         }
370         catch (Exception e)
371         {
372             Log.e(LIVE_CARD_TAG,"Failed to get sensor values",e);
373             return null;
374         }
375     }

376

377     // Get the new data points that we will need to graph
378
379     /** This function computes the points that have to be plotted
380      *
381      * @param sensor , the name of sensor (pH or Temperature)
382      * @param last_timestamp , the previous value of timestamp
383      * @return JSONArray, return a list of values that corresponds to the points that have to be plotted
384      */
385     private JSONArray getDataPoints(String sensor, double last_timestamp)
386     {
387         try
388         {
389             String url = "http://planar-contact-601.appspot.com/graphing_data?sensor=" + sensor + "&last_timestamp=" + last_timestamp;
390             String values = getURL(url);
391             return new JSONArray(new JSONTokener(values));
392         }
393         catch (Exception e)
394         {
395             Log.e(LIVE_CARD_TAG,"Failed to get data points",e);
396             return null;
397         }
398     }

399

400     /** This method gets data from the given url website
401      *
402      * @param _url, url in which the data are contained
403      * @return String, contained data from the given url
```



```

455         double timestamp = (Double) point.get("timestamp");
456         double value = (Double) point.get("value");
457         curr_data.add(new DataPoint(timestamp, value));
458     } catch (JSONException e) {
459         Log.e(LIVE_CARD_TAG, "JSON error", e);
460         //continue;
461     }
462 }
463
464 // Clear out points that are over an hour old
465 while (true) {
466     if (curr_data.size() > 0 && curr_data.get(0).getTimestamp() < (curr_data.get(0).getTimestamp() - 3600000))
467         Log.i(LIVE_CARD_TAG, "Deleting old data point");
468     curr_data.remove(0);
469 } else {
470     break;
471 }
472
473 // If there are no points don't show a graph
474 if (curr_data.size() == 0) {
475     continue;
476 }
477
478 // Store the timestamps and values in separate arrays for graphing
479 ArrayList<Double> timestamps = new ArrayList<Double>();
480 ArrayList<Double> values = new ArrayList<Double>();
481 for (DataPoint curr_point : curr_data) {
482     timestamps.add(curr_point.getTimestamp());
483     values.add(curr_point.getValue());
484     //System.out.println("Value = " + values.get(j));
485 }
486
487 mSensorAverage.put(curr_sensor, computeAverage(values));
488
489 // Scale the timestamp data
490 double maxTimestamp = Collections.max(timestamps);
491 double minTimestamp = Collections.min(timestamps);
492 maxTimestamp *= 1.2;
493 minTimestamp *= 0.8;
494 double intervalSize = maxTimestamp - minTimestamp;
495 for (int i1 = 0; i1 < timestamps.size(); i1++) {
496     double currTimestamp = timestamps.get(i1);
497     currTimestamp -= minTimestamp;
498     currTimestamp /= intervalSize;
499     currTimestamp *= 100;
500     timestamps.set(i1, currTimestamp);
501 }
502
503 // Scale the value data
504 double maxVal = Collections.max(values);
505 double minVal = Collections.min(values);
506 maxVal *= 1.2;
507 minVal *= 0.8;

```

```

503         intervalSize = maxVal - minVal;
504         for (int i1 = 0; i1 < values.size(); i1++) {
505             double currVal = values.get(i1);
506             currVal -= minVal;
507             currVal /= intervalSize;
508             currVal *= 100;
509             values.set(i1, currVal);
510         }
511     }
512
513     // Data xData = Data.newData(timestamps);
514     Data yData = Data.newData(values);
515
516
517     Plot plot = Plots.newPlot(yData);
518     LineChart lineChart = GCharts.newLineChart(plot);
519     lineChart.setSize(400, 200);
520     lineChart.addYAxisLabels(AxisLabelsFactory.newNumericRangeAxisLabels(minVal, maxVal));
521
522
523     mCurrentSensorGraphs.put(curr_sensor, getBitmapFromURL(lineChart.toURLString()));
524
525 }
526
527 mCallback.SetSensorAvg(mSensorAverage);
528 mCallback.setSensorGraphs(mCurrentSensorGraphs);
529 Log.i("Main Service", "Graphs updated");
530
531     // restart it after a given amount of time
532     mHandler.postDelayed(mUpdateSensorGraphsRunnable, GRAPH_UPDATE_DELAY_MILLIS);
533 }
534
535
536     private double computeAverage(ArrayList<Double> values) {
537         double sum = 0.0;
538         if(!values.isEmpty()){
539             for(Double value : values){
540                 sum += value;
541             }
542             return sum/values.size();
543         }
544         return sum;
545     }
546
547     /**
548      * @return mIsStopped, true if the runnable is stopped, false otherwise
549      */
550
551     public boolean isStopped()

```

```
553     {
554         return mIsStopped;
555     }
556
557     /**
558      * @param isStopped, true if the user wishes to stop the runnable, false otherwise
559      */
560     public void setStop(boolean isStopped)
561     {
562         this.mIsStopped = isStopped;
563     }
564 }
565 //
566 //
567 //    class DataPoint {
568 //        private final double mTimestamp;
569 //        private final double mValue;
570 //        DataPoint(double timestamp, double value) {
571 //            mTimestamp = timestamp;
572 //            mValue = value;
573 //        }
574 //        public double getTimestamp() {
575 //            return mTimestamp;
576 //        }
577 //        public double getValue() {
578 //            return mValue;
579 //        }
580 //    }
581
582
583     /**
584      * Runnable that implements the task for downloading beating image
585      */
586     public class ImageDownloader implements Runnable {
587         private String url;
588         private Context c;
589         private boolean mIsStopped = false;
590
591         /**
592          * @param url, url link of image
593          * @param c, is the context in which the request of downloading image has been sent
594          * @see
595          */
596         public ImageDownloader(String url, Context c)
597         {
598             this.url = url;
599             this.c = c;
600         }
601     }
602 }
```

```
601     /** It implements the task of ImageDownloader runnable
602      *
603      * @see
604      */
605     @Override
606     public void run()
607     {
608         if(!isStopped())
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 runnable
618      *
619      * @return mIsStopped, true if the runnable is stopped, false otherwise
620      */
621     public boolean isStopped()
622     {
623         return mIsStopped;
624     }
625
626     /** It is the setter that allows to stop the runnable
627      *
628      * @param isStopped, true if the user wishes to stop the runnable, false otherwise
629      */
630     public void setIsStopped(boolean isStopped)
631     {
632         this.mIsStopped = isStopped;
633     }
634
635     /** This method pulls an image from the given url
636      *
637      * @param urlLink where the image is stored
638      * @return Bitmap which contains the image of the graph
639      * @throws java.io.IOException if an IO exception occurred during the download
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();
```

```

        Log.i(BEATING_TAG, "download completed in "
                + ((System.currentTimeMillis() - startTime) / 1000)
                + " sec");
    return BitmapFactory.decodeStream(inputStream);
} catch (IOException e) {
    e.printStackTrace();
    Log.e(BEATING_TAG, e.getMessage());
    return null;
}
}

//=====
// =====

// Runnable that updates the microscope video
private class UpdateMicroscopeVideoRunnable implements Runnable {
    private boolean mIsStopped = false;
    public void run() {
        if (!mIsStopped) {
            getMicroscopeVideo();
            mHandler.postDelayed(mUpdateMicroscopeVideoRunnable, VIDEO_UPDATE_DELAY_MILLIS);
        }
    }
    public boolean isStopped() {
        return mIsStopped;
    }
    public void setStop(boolean isStopped) {
        this.mIsStopped = isStopped;
    }
}

// Pull the microscope video from a URL
private void getMicroscopeVideo() {
    try {
        URL url = new URL("http://planar-contact-601.appspot.com/video/view");
        long startTime = System.currentTimeMillis();
        Log.i(VIDEO_TAG, "video download beginning: "+url);
        URLConnection ucon = url.openConnection();
        ucon.setReadTimeout(0);
        ucon.setConnectTimeout(0);
        // Define InputStreams to read from the URLConnection.
        InputStream is = ucon.getInputStream();
        BufferedInputStream inStream = new BufferedInputStream(is, 1024*5);
        File file = new File(TEMP_VIDEO_FILE_NAME);

        FileOutputStream outStream = new FileOutputStream(file);
        FileLock lock = outStream.getChannel().lock();
    }
}

```

```

699         byte[] buff = new byte[1024*5];
700         // Read bytes (and store them) until there is nothing more to read(-1)
701         int len;
702         while ((len = inStream.read(buff)) != -1) {
703             outStream.write(buff,0,len);
704         }
705         // Clean up
706
707         outStream.flush();
708         lock.release();
709         outStream.close();
710         inStream.close();
711         Log.i(VIDEO_TAG, "download completed in "
712               + ((System.currentTimeMillis() - startTime) / 1000)
713               + " sec");
714     }
715     catch (IOException e) {
716         Log.e(VIDEO_TAG, "Failed to download microscope video", e);
717     }
718 }
719
720 }
721 }
```

Listing A.18: AppDrawer.java

```

1 package com.google.android.glass.sample.klabinterface;
2
3 import com.google.android.glass.timeline.DirectRenderingCallback;
4 import com.jjoe64.graphview.GraphView;
5
6 import android.content.Context;
7 import android.content.Intent;
8 import android.graphics.Bitmap;
9 import android.graphics.Canvas;
10 import android.os.Environment;
11 import android.os.Handler;
12 import android.os.SystemClock;
13 import android.util.Log;
14 import android.view.SurfaceHolder;
15 import android.view.View;
16
17 import java.util.Map;
18
19 /**
20 * {@link DirectRenderingCallback} used to draw the chronometer on the timeline {@link com.google.an
21 * Rendering requires that:
22 * <ol>
23 * <li>a {@link SurfaceHolder} has been created through monitoring the
```

```

*      {@link SurfaceHolder.Callback#(SurfaceHolder)} and
25     *      {@link SurfaceHolder.Callback#(SurfaceHolder)} callbacks.
* <li>rendering has not been paused (defaults to rendering) through monitoring the
27     *      {@link com.google.android.glass.timeline.DirectRenderingCallback#renderingPaused(SurfaceHolder)}
* </ol>
29     * As this class uses an inflated {@link View} to draw on the {@link SurfaceHolder}'s
* {@link Canvas}, monitoring the
31     * {@link SurfaceHolder.Callback#(SurfaceHolder, int, int, int)} callback is also
* required to properly measure and layout the {@link View}'s dimension.
33 */
34
35     public class AppDrawer implements DirectRenderingCallback {
36
37         /** INT associated to the Menu view request */
38         private static final int MENU = 0;
39
40         /** INT associated to the PH view request */
41         private static final int PH = 1;
42
43         /** INT associated to the Menu view request */
44         private static final int TEMPERATURE = 2;
45
46         /** INT associated to the Video view request */
47         private static final int VIDEO = 3;
48
49         /** INT associated to the Beating view request */
50         private static final int BEATING = 4;
51
52
53         private VideoThread mRenderThread;
54
55
56         /** Bitmap in which the beating image is stored */
57         private Bitmap bmp;
58
59         private boolean mReady;
60
61
62         private static final String TAG = AppDrawer..class.getSimpleName();
63
64
65         private final MainView mMainView;
66         private final BeatingView mBeatingView;
67
68         /** View object of the GlassWear pH window */
69         private final PHViewer mPhViewer;
70
71         private final TemperatureView mTemperatureView;
72
73
74         private SurfaceHolder mHolder;
75
76         private boolean mRenderingPaused;
77
78
79         /** Hash table in which the sensors graphs are stored */
80         private Map<String, Bitmap> mCurrentSensorGraphs;
81
82         /** Hash table in which the sensors values are stored */
83         private Map<String, Double> mCurrentSensorValues;
84
85         private Map<String, Double> mSensorAverage;
86
87         private GraphView mGraphView;
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
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
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
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
636
637
637
638
639
639
640
641
642
643
644
645
646
647
647
648
649
649
650
651
652
653
654
655
656
657
657
658
659
659
660
661
662
663
664
665
666
667
667
668
669
669
670
671
672
673
674
675
676
676
677
678
678
679
679
680
681
682
683
684
685
686
686
687
688
688
689
689
690
691
692
693
694
695
696
696
697
698
698
699
699
700
701
702
703
704
705
706
707
707
708
709
709
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560

```

```
73     private final MainView.Listener mMainListener = new MainView.Listener() {
75
77         @Override
78         public void onChange() {
79             // mMainDone = true;
80             //mBeatingView.setBaseMillis(0);
81             updateRenderingState();
82         }
83     };
84
85     /**
86      * Defines the Listener of pH viewer, it is used to communicate with
87      * that viewer and allowing its viewing */
88     private final PHViewer.Listener mPhListener = new PHViewer.Listener(){
89         @Override
90         /* This function is used when the {@link com.example.alik.bwhglass.PHViewer} Class wants
91         * to change the view object.
92         */
93         public void onChange(){
94             //state = PH;
95             updateRenderingState();
96         }
97     };
98
99     private final BeatingView.Listener mBeatingListener = new BeatingView.Listener() {
100
101         @Override
102         public void onChange() {
103             updateRenderingState();
104         }
105     };
106
107     private final TemperatureView.Listener mTemperatureListener = new TemperatureView.Listener() {
108
109         @Override
110         public void onChange() {
111             updateRenderingState();
112         }
113     };
114
115     /**
116      * Defines the ManageBitmap of Beating viewer, it is used to communicate with
117      * that viewer in order to update the bitmap to be displayed*/
118     private final BeatingView.ManageBitmap mBeatingBitmap = new BeatingView.ManageBitmap(){
119
120         @Override
121         /* Getter that returns the bitmap of the beating image to be displayed
122         */
```

```

    * @return Bitmap, sensor values with the values of pH and Temperature
123   */
124
125   public Bitmap getBitmap() {
126
126     return bmp;
127   }
128
129
130   public boolean isReady(){
131
132     return mReady;
133   }
134
135 }

136 /**
137  * Defines the ManageDataGraph of PH viewer, it is used to communicate with
138  * that viewer in order to update the hash map which contains the graph to be displayed*/
139 private final PHViewer.ManageBitmap mPHManageDataGraph = new PHViewer.ManageBitmap(){

140
141   public Bitmap getBitmap() {
142
143     return mCurrentSensorGraphs.get("pH");
144   }
145
146
147   public double getAvg(){return mSensorAverage.get("pH");}
148
149
150   public boolean isReady(){
151
152     return mReady;
153   }
154
155 }

156 /**
157  * Defines the ManageDataGraph of PH viewer, it is used to communicate with
158  * that viewer in order to update the hash map which contains the graph to be displayed*/
159 private final TemperatureView.ManageBitmap mTemperatureManageDataGraph = new TemperatureView.ManageBitmap(){

160
161   public Bitmap getBitmap() {
162
163     return mCurrentSensorGraphs.get("pH");
164   }
165
166
167   public double getAvg(){return mSensorAverage.get("Temperature");}
168
169
170   public boolean isReady(){
171
172     return mReady;
173   }
174
175 }

176
177
178   public AppDrawer.(Context context) {
179
180     this(new MainView(context), new BeatingView(context), new PHViewer(context), new TemperatureView(context));
181
182
183   public AppDrawer.(MainView countDownView, BeatingView chronometerView, PHViewer phViewer, TemperatureView temperatureView) {
184
185     mMainView = countDownView;
186
187     mMainView.setListener(mMainListener);
188
189   }
190
191
192   public void setMainView(MainView mainView) {
193
194     mMainView = mainView;
195
196     mMainView.setListener(mMainListener);
197
198   }
199
200
201   public void setBeatingView(BeatingView beatingView) {
202
203     mBeatingView = beatingView;
204
205     mBeatingView.setListener(mBeatingListener);
206
207   }
208
209
210   public void setPHViewer(PHViewer phViewer) {
211
212     mPHViewer = phViewer;
213
214     mPHViewer.setListener(mPHListener);
215
216   }
217
218
219   public void setTemperatureView(TemperatureView temperatureView) {
220
221     mTemperatureView = temperatureView;
222
223     mTemperatureView.setListener(mTemperatureListener);
224
225   }
226
227
228   public void setChronometerView(ChronometerView chronometerView) {
229
230     mChronometerView = chronometerView;
231
232     mChronometerView.setListener(mChronometerListener);
233
234   }
235
236
237   public void setCountDownView(CountDownView countDownView) {
238
239     mCountDownView = countDownView;
240
241     mCountDownView.setListener(mCountDownListener);
242
243   }
244
245
246   public void setListner(Listener listener) {
247
248     mListener = listener;
249
250   }
251
252
253   public void setMainListener(MainListener mainListener) {
254
255     mMainListener = mainListener;
256
257   }
258
259
260   public void setBeatingListener(BeatingListener beatingListener) {
261
262     mBeatingListener = beatingListener;
263
264   }
265
266
267   public void setPHListener(PHListener phListener) {
268
269     mPHListener = phListener;
270
271   }
272
273
274   public void setTemperatureListener(TemperatureListener temperatureListener) {
275
276     mTemperatureListener = temperatureListener;
277
278   }
279
280
281   public void setChronometerListener(ChronometerListener chronometerListener) {
282
283     mChronometerListener = chronometerListener;
284
285   }
286
287
288   public void setCountDownListener(CountDownListener countDownListener) {
289
290     mCountDownListener = countDownListener;
291
292   }
293
294
295   public void setChronometer(Chronometer chronometer) {
296
297     mChronometer = chronometer;
298
299   }
299
300
301   public void setCountDown(CountDown countDown) {
302
303     mCountDown = countDown;
304
305   }
306
307
308   public void setMain(Main main) {
309
310     mMain = main;
311
312   }
313
314
315   public void setBeating(Beating beating) {
316
317     mBeating = beating;
318
319   }
320
321
322   public void setPH(PH ph) {
323
324     mPH = ph;
325
326   }
327
328
329   public void setTemperature(Temperature temperature) {
330
331     mTemperature = temperature;
332
333   }
334
335
336   public void setCountDown(CountDown countDown) {
337
338     mCountDown = countDown;
339
340   }
341
342
343   public void setChronometer(Chronometer chronometer) {
344
345     mChronometer = chronometer;
346
347   }
348
349
350   public void setMain(Main main) {
351
352     mMain = main;
353
354   }
355
356
357   public void setBeating(Beating beating) {
358
359     mBeating = beating;
360
361   }
362
363
364   public void setPH(PH ph) {
365
366     mPH = ph;
367
368   }
369
370
371   public void setTemperature(Temperature temperature) {
372
373     mTemperature = temperature;
374
375   }
376
377
378   public void setCountDown(CountDown countDown) {
379
380     mCountDown = countDown;
381
382   }
383
384
385   public void setChronometer(Chronometer chronometer) {
386
387     mChronometer = chronometer;
388
389   }
390
391
392   public void setMain(Main main) {
393
394     mMain = main;
395
396   }
397
398
399   public void setBeating(Beating beating) {
400
401     mBeating = beating;
402
403   }
404
405
406   public void setPH(PH ph) {
407
408     mPH = ph;
409
410   }
411
412
413   public void setTemperature(Temperature temperature) {
414
415     mTemperature = temperature;
416
417   }
418
419
420   public void setCountDown(CountDown countDown) {
421
422     mCountDown = countDown;
423
424   }
425
426
427   public void setChronometer(Chronometer chronometer) {
428
429     mChronometer = chronometer;
430
431   }
432
433
434   public void setMain(Main main) {
435
436     mMain = main;
437
438   }
439
440
441   public void setBeating(Beating beating) {
442
443     mBeating = beating;
444
445   }
446
447
448   public void setPH(PH ph) {
449
450     mPH = ph;
451
452   }
453
454
455   public void setTemperature(Temperature temperature) {
456
457     mTemperature = temperature;
458
459   }
460
461
462   public void setCountDown(CountDown countDown) {
463
464     mCountDown = countDown;
465
466   }
467
468
469   public void setChronometer(Chronometer chronometer) {
470
471     mChronometer = chronometer;
472
473   }
474
475
476   public void setMain(Main main) {
477
478     mMain = main;
479
480   }
481
482
483   public void setBeating(Beating beating) {
484
485     mBeating = beating;
486
487   }
488
489
490   public void setPH(PH ph) {
491
492     mPH = ph;
493
494   }
495
496
497   public void setTemperature(Temperature temperature) {
498
499     mTemperature = temperature;
500
501   }
502
503
504   public void setCountDown(CountDown countDown) {
505
506     mCountDown = countDown;
507
508   }
509
510
511   public void setChronometer(Chronometer chronometer) {
512
513     mChronometer = chronometer;
514
515   }
516
517
518   public void setMain(Main main) {
519
520     mMain = main;
521
522   }
523
524
525   public void setBeating(Beating beating) {
526
527     mBeating = beating;
528
529   }
530
531
532   public void setPH(PH ph) {
533
534     mPH = ph;
535
536   }
537
538
539   public void setTemperature(Temperature temperature) {
540
541     mTemperature = temperature;
542
543   }
544
545
546   public void setCountDown(CountDown countDown) {
547
548     mCountDown = countDown;
549
550   }
551
552
553   public void setChronometer(Chronometer chronometer) {
554
555     mChronometer = chronometer;
556
557   }
558
559
560   public void setMain(Main main) {
561
562     mMain = main;
563
564   }
565
566
567   public void setBeating(Beating beating) {
568
569     mBeating = beating;
570
571   }
572
573
574   public void setPH(PH ph) {
575
576     mPH = ph;
577
578   }
579
580
581   public void setTemperature(Temperature temperature) {
582
583     mTemperature = temperature;
584
585   }
586
587
588   public void setCountDown(CountDown countDown) {
589
590     mCountDown = countDown;
591
592   }
593
594
595   public void setChronometer(Chronometer chronometer) {
596
597     mChronometer = chronometer;
598
599   }
599
600
601   public void setMain(Main main) {
602
603     mMain = main;
604
605   }
606
607
608   public void setBeating(Beating beating) {
609
610     mBeating = beating;
611
612   }
613
614
615   public void setPH(PH ph) {
616
617     mPH = ph;
618
619   }
620
621
622   public void setTemperature(Temperature temperature) {
623
624     mTemperature = temperature;
625
626   }
627
628
629   public void setCountDown(CountDown countDown) {
630
631     mCountDown = countDown;
632
633   }
634
635
636   public void setChronometer(Chronometer chronometer) {
637
638     mChronometer = chronometer;
639
640   }
641
642
643   public void setMain(Main main) {
644
645     mMain = main;
646
647   }
648
649
650   public void setBeating(Beating beating) {
651
652     mBeating = beating;
653
654   }
655
656
657   public void setPH(PH ph) {
658
659     mPH = ph;
660
661   }
662
663
664   public void setTemperature(Temperature temperature) {
665
666     mTemperature = temperature;
667
668   }
669
670
671   public void setCountDown(CountDown countDown) {
672
673     mCountDown = countDown;
674
675   }
676
677
678   public void setChronometer(Chronometer chronometer) {
679
680     mChronometer = chronometer;
681
682   }
683
684
685   public void setMain(Main main) {
686
687     mMain = main;
688
689   }
690
691
692   public void setBeating(Beating beating) {
693
694     mBeating = beating;
695
696   }
697
698
699   public void setPH(PH ph) {
700
701     mPH = ph;
702
703   }
704
705
706   public void setTemperature(Temperature temperature) {
707
708     mTemperature = temperature;
709
710   }
711
712
713   public void setCountDown(CountDown countDown) {
714
715     mCountDown = countDown;
716
717   }
718
719
720   public void setChronometer(Chronometer chronometer) {
721
722     mChronometer = chronometer;
723
724   }
725
726
727   public void setMain(Main main) {
728
729     mMain = main;
730
731   }
732
733
734   public void setBeating(Beating beating) {
735
736     mBeating = beating;
737
738   }
739
740
741   public void setPH(PH ph) {
742
743     mPH = ph;
744
745   }
746
747
748   public void setTemperature(Temperature temperature) {
749
750     mTemperature = temperature;
751
752   }
753
754
755   public void setCountDown(CountDown countDown) {
756
757     mCountDown = countDown;
758
759   }
760
761
762   public void setChronometer(Chronometer chronometer) {
763
764     mChronometer = chronometer;
765
766   }
767
768
769   public void setMain(Main main) {
770
771     mMain = main;
772
773   }
774
775
776   public void setBeating(Beating beating) {
777
778     mBeating = beating;
779
780   }
781
782
783   public void setPH(PH ph) {
784
785     mPH = ph;
786
787   }
788
789
790   public void setTemperature(Temperature temperature) {
791
792     mTemperature = temperature;
793
794   }
795
796
797   public void setCountDown(CountDown countDown) {
798
799     mCountDown = countDown;
800
801   }
802
803
804   public void setChronometer(Chronometer chronometer) {
805
806     mChronometer = chronometer;
807
808   }
809
810
811   public void setMain(Main main) {
812
813     mMain = main;
814
815   }
816
817
818   public void setBeating(Beating beating) {
819
820     mBeating = beating;
821
822   }
823
824
825   public void setPH(PH ph) {
826
827     mPH = ph;
828
829   }
830
831
832   public void setTemperature(Temperature temperature) {
833
834     mTemperature = temperature;
835
836   }
837
838
839   public void setCountDown(CountDown countDown) {
840
841     mCountDown = countDown;
842
843   }
844
845
846   public void setChronometer(Chronometer chronometer) {
847
848     mChronometer = chronometer;
849
850   }
851
852
853   public void setMain(Main main) {
854
855     mMain = main;
856
857   }
858
859
860   public void setBeating(Beating beating) {
861
862     mBeating = beating;
863
864   }
865
866
867   public void setPH(PH ph) {
868
869     mPH = ph;
870
871   }
872
873
874   public void setTemperature(Temperature temperature) {
875
876     mTemperature = temperature;
877
878   }
879
880
881   public void setCountDown(CountDown countDown) {
882
883     mCountDown = countDown;
884
885   }
886
887
888   public void setChronometer(Chronometer chronometer) {
889
890     mChronometer = chronometer;
891
892   }
893
894
895   public void setMain(Main main) {
896
897     mMain = main;
898
899   }
900
901
902   public void setBeating(Beating beating) {
903
904     mBeating = beating;
905
906   }
907
908
909   public void setPH(PH ph) {
910
911     mPH = ph;
912
913   }
914
915
916   public void setTemperature(Temperature temperature) {
917
918     mTemperature = temperature;
919
920   }
921
922
923   public void setCountDown(CountDown countDown) {
924
925     mCountDown = countDown;
926
927   }
928
929
930   public void setChronometer(Chronometer chronometer) {
931
932     mChronometer = chronometer;
933
934   }
935
936
937   public void setMain(Main main) {
938
939     mMain = main;
940
941   }
942
943
944   public void setBeating(Beating beating) {
945
946     mBeating = beating;
947
948   }
949
950
951   public void setPH(PH ph) {
952
953     mPH = ph;
954
955   }
956
957
958   public void setTemperature(Temperature temperature) {
959
960     mTemperature = temperature;
961
962   }
963
964
965   public void setCountDown(CountDown countDown) {
966
967     mCountDown = countDown;
968
969   }
970
971
972   public void setChronometer(Chronometer chronometer) {
973
974     mChronometer = chronometer;
975
976   }
977
978
979   public void setMain(Main main) {
980
981     mMain = main;
982
983   }
984
985
986   public void setBeating(Beating beating) {
987
988     mBeating = beating;
989
990   }
991
992
993   public void setPH(PH ph) {
994
995     mPH = ph;
996
997   }
998
999
1000   public void setTemperature(Temperature temperature) {
1001
1002     mTemperature = temperature;
1003
1004   }
1005
1006
1007   public void setCountDown(CountDown countDown) {
1008
1009     mCountDown = countDown;
1010
1011   }
1012
1013
1014   public void setChronometer(Chronometer chronometer) {
1015
1016     mChronometer = chronometer;
1017
1018   }
1019
1020
1021   public void setMain(Main main) {
1022
1023     mMain = main;
1024
1025   }
1026
1027
1028   public void setBeating(Beating beating) {
1029
1030     mBeating = beating;
1031
1032   }
1033
1034
1035   public void setPH(PH ph) {
1036
1037     mPH = ph;
1038
1039   }
1040
1041
1042   public void setTemperature(Temperature temperature) {
1043
1044     mTemperature = temperature;
1045
1046   }
1047
1048
1049   public void setCountDown(CountDown countDown) {
1050
1051     mCountDown = countDown;
1052
1053   }
1054
1055
1056   public void setChronometer(Chronometer chronometer) {
1057
1058     mChronometer = chronometer;
1059
1060   }
1061
1062
1063   public void setMain(Main main) {
1064
1065     mMain = main;
1066
1067   }
1068
1069
1070   public void setBeating(Beating beating) {
1071
1072     mBeating = beating;
1073
1074   }
1075
1076
1077   public void setPH(PH ph) {
1078
1079     mPH = ph;
1080
1081   }
1082
1083
1084   public void setTemperature(Temperature temperature) {
1085
1086     mTemperature = temperature;
1087
1088   }
1089
1090
1091   public void setCountDown(CountDown countDown) {
1092
1093     mCountDown = countDown;
1094
1095   }
1096
1097
1098   public void setChronometer(Chronometer chronometer) {
1099
1100     mChronometer = chronometer;
1101
1102   }
1103
1104
1105   public void setMain(Main main) {
1106
1107     mMain = main;
1108
1109   }
1110
1111
1112   public void setBeating(Beating beating) {
1113
1114     mBeating = beating;
1115
1116   }
1117
1118
1119   public void setPH(PH ph) {
1120
1121     mPH = ph;
1122
1123   }
1124
1125
1126   public void setTemperature(Temperature temperature) {
1127
1128     mTemperature = temperature;
1129
1130   }
1131
1132
1133   public void setCountDown(CountDown countDown) {
1134
1135     mCountDown = countDown;
1136
1137   }
1138
1139
1140   public void setChronometer(Chronometer chronometer) {
1141
1142     mChronometer = chronometer;
1143
1144   }
1145
1146
1147   public void setMain(Main main) {
1148
1149     mMain = main;
1150
1151   }
1152
1153
1154   public void setBeating(Beating beating) {
1155
1156     mBeating = beating;
1157
1158   }
1159
1160
1161   public void setPH(PH ph) {
1162
1163     mPH = ph;
1164
1165   }
1166
1167
1168   public void setTemperature(Temperature temperature) {
1169
1170     mTemperature = temperature;
1171
1172   }
1173
1174
1175   public void setCountDown(CountDown countDown) {
1176
1177     mCountDown = countDown;
1178
1179   }
1180
1181
1182   public void setChronometer(Chronometer chronometer) {
1183
1184     mChronometer = chronometer;
1185
1186   }
1187
1188
1189   public void setMain(Main main) {
1190
1191     mMain = main;
1192
1193   }
1194
1195
1196   public void setBeating(Beating beating) {
1197
1198     mBeating = beating;
1199
1200   }
1201
1202
1203   public void setPH(PH ph) {
1204
1205     mPH = ph;
1206
1207   }
1208
1209
1210   public void setTemperature(Temperature temperature) {
1211
1212     mTemperature = temperature;
1213
1214   }
1215
1216
1217   public void setCountDown(CountDown countDown) {
1218
1219     mCountDown = countDown;
1220
1221   }
1222
1223
1224   public void setChronometer(Chronometer chronometer) {
1225
1226     mChronometer = chronometer;
1227
1228   }
1229
1230
1231   public void setMain(Main main) {
1232
1233     mMain = main;
1234
1235   }
1236
1237
1238   public void setBeating(Beating beating) {
1239
1240     mBeating = beating;
1241
1242   }
1243
1244
1245   public void setPH(PH ph) {
1246
1247     mPH = ph;
1248
1249   }
1250
1251
1252   public void setTemperature(Temperature temperature) {
1253
1254     mTemperature = temperature;
1255
1256   }
1257
1258
1259   public void setCountDown(CountDown countDown) {
1260
1261     mCountDown = countDown;
1262
1263   }
1264
1265
1266   public void setChronometer(Chronometer chronometer) {
1267
1268     mChronometer = chronometer;
1269
1270   }
1271
1272
1273   public void setMain(Main main) {
1274
1275     mMain = main;
1276
1277   }
1278
1279
1280   public void setBeating(Beating beating) {
1281
1282     mBeating = beating;
1283
1284   }
1285
1286
1287   public void setPH(PH ph) {
1288
1289     mPH = ph;
1290
1291   }
1292
1293
1294   public void setTemperature(Temperature temperature) {
1295
1296     mTemperature = temperature;
1297
1298   }
1299
1300
1301   public void setCountDown(CountDown countDown) {
1302
1303     mCountDown = countDown;
1304
1305   }
1306
1307
1308   public void setChronometer(Chronometer chronometer) {
1309
1310     mChronometer = chronometer;
1311
1312   }
1313
1314
1315   public void setMain(Main main) {
1316
1317     mMain = main;
1318
1319   }
1320
1321
1322   public void setBeating(Beating beating) {
1323
1324     mBeating = beating;
1325
1326   }
1327
1328
1329   public void setPH(PH ph) {
1330
1331     mPH = ph;
1332
1333   }
1334
1335
1336   public void setTemperature(Temperature temperature) {
1337
1338     mTemperature = temperature;
1339
1340   }
1341
1342
1343   public void setCountDown(CountDown countDown) {
1344
1345     mCountDown = countDown;
1346
1347   }
1348
1349
1350   public void setChronometer(Chronometer chronometer) {
1351
1352     mChronometer = chronometer;
1353
1354   }
1355
1356
1357   public void setMain(Main main) {
1358
1359     mMain = main;
1360
1361   }
1362
1363
1364   public void setBeating(Beating beating) {
1365
1366     mBeating = beating;
1367
1368   }
1369
1370
1371   public void setPH(PH ph) {
1372
1373     mPH = ph;
1374
1375   }
1376
1377
1378   public void setTemperature(Temperature temperature) {
1379
1380     mTemperature = temperature;
1381
1382   }
1383
1384
1385   public void setCountDown(CountDown countDown) {
1386
1387     mCountDown = countDown;
1388
1389   }
1390
1391
1392   public void setChronometer(Chronometer chronometer) {
1393
1394     mChronometer = chronometer;
1395
1396   }
1397
1398
1399   public void setMain(Main main) {
1400
1401     mMain = main;
1402
1403   }
1404
1405
1406   public void setBeating(Beating beating) {
1407
1408     mBeating = beating;
1409
1410   }
1411
1412
1413   public void setPH(PH ph) {
1414
1415     mPH = ph;
1416
1417   }
1418
1419
1420   public void setTemperature(Temperature temperature) {
1421
1422     mTemperature = temperature;
1423
1424   }
1425
1426
1427   public void setCountDown(CountDown countDown) {
1428
1429     mCountDown = countDown;
1430
1431   }
1432
1433
1434   public void setChronometer(Chronometer chronometer) {
1435
1436     mChronometer = chronometer;
1437
1438   }
1439
1440
1441   public void setMain(Main main) {
1442
1443     mMain = main;
1444
1445   }
1446
1447
1448   public void setBeating(Beating beating) {
1449
1450     mBeating = beating;
1451
1452   }
1453
1454
1455   public void setPH(PH ph) {
1456
1457     mPH = ph;
1458
1459   }
1460
1461
1462   public void setTemperature(Temperature temperature) {
1463
1464     mTemperature = temperature;
1465
1466   }
1467
1468
1469   public void setCountDown(CountDown countDown) {
1470
1471     mCountDown = countDown;
1472
1473   }
1474
1475
1476   public void setChronometer(Chronometer chronometer) {
1477
1478     mChronometer = chronometer;
1479
1480   }
1481
1482
1483   public void setMain(Main main) {
1484
1485     mMain = main;
1486
1487   }
1488
1489
1490   public void setBeating(Beating beating) {
1491
1492     mBeating = beating;
1493
1494   }
1495
1496
1497   public void setPH(PH ph) {
1498
1499     mPH = ph;
1500
1501   }
1502
1503
1504   public void setTemperature(Temperature temperature) {
1505
1506     mTemperature = temperature;
1507
1508   }
1509
1510
1511   public void setCountDown(CountDown countDown) {
1512
1513     mCountDown = countDown;
1514
1515   }
1516
1517
1518   public void setChronometer(Chronometer chronometer) {
1519
1520     mChronometer = chronometer;
1521
1522   }
1523
1524
1525   public void setMain(Main main) {
1526
1527     mMain = main;
1528
1529   }
1530
1531
1532   public void setBeating(Beating beating) {
1533
1534     mBeating = beating;
1535
1536   }
1537
1538
1539   public void setPH(PH ph) {
1540
1541     mPH = ph;
1542
1543   }
1544
1545
1546   public void setTemperature(Temperature temperature) {
1547
1548     mTemperature = temperature;
1549
1550   }
1551
1552
1553   public void setCountDown(CountDown countDown) {
1554
1555     mCountDown = countDown;
1556
1557   }
1558
1559
1560   public void setChronometer(Chronometer chronometer) {
1561
1562     mChronometer = chronometer;
1563
1564   }
1565
1566
1567   public void setMain(Main main) {
1568
1569     mMain = main;
1570
1571   }
1572
1573
1574   public void setBeating(Beating beating) {
1575
1576     mBeating = beating;
1577
1578   }
1579
1580
1581   public void setPH(PH ph) {
1582
1583     mPH = ph;
1584
1585   }
1586
1587
1588   public void setTemperature(Temperature temperature) {
1589
1590     mTemperature = temperature;
1591
1592   }
1593
1594
1595   public void setCountDown(CountDown countDown) {
1596
1597     mCountDown = countDown;
1598
1599   }
1600
1601
1602   public void setChronometer(Chronometer chronometer) {
1603
1604     mChronometer = chronometer;
1605
1606   }
1607
1608
1609   public void setMain(Main main) {
1610
1611     mMain = main;
1612
1613   }
1614
1615
1616   public void setBeating(Beating beating) {
1617
1618     mBeating = beating;
1619
1620   }
1621
1622
1623   public void setPH(PH ph) {
1624
1625     mPH = ph;
1626
1627   }
1628
1629
1630   public void setTemperature(Temperature temperature) {
1631
1632     mTemperature = temperature;
1633
1634   }
1635
1636
1637   public void setCountDown(CountDown countDown) {
1638
1639     mCountDown = countDown;
1640
1641   }
1642
1643
1644   public void setChronometer(Chronometer chronometer) {
1645
1646     mChronometer = chronometer;
1647
1648   }
1649
1650
1651   public void setMain(Main main) {
1652
1653     mMain = main;
1654
1655   }
1656
1657
1658   public void setBeating(Beating beating) {
1659
1660     mBeating = beating;
1661
1662   }
1663
1664
1665   public void setPH(PH ph) {
1666
1667     mPH = ph;
1668
1669   }
1670
1671
1672   public void setTemperature(Temperature temperature) {
1673
1674     mTemperature = temperature;
1675
1676   }
1677
1678
1679   public void setCountDown(CountDown countDown) {
1680
1681     mCountDown = countDown;
1682
1683   }
1684
1685
1686   public void setChronometer(Chronometer chronometer) {
1687
1688     mChronometer = chronometer;
1689
1690   }
1691
1692
1693   public void setMain(Main main) {
1694
1695     mMain = main;
1696
1697   }
1698
1699
1700   public void setBeating(Beating beating) {
1701
1702     mBeating = beating;
1703
1704   }
1705

```

```
171     mBeatingView = chronometerView;
173     mBeatingView.setListener(mBeatingListener);
175     mBeatingView.setManageBPM(mBeatingBitmap);
176
177     mPhViewer = phViewer;
178     mPhViewer.setListener(mPhListener);
179     mPhViewer.setManageBPM(mPHManageDataGraph);
180
181     mTemperatureView = temperatureView;
182     mTemperatureView.setListener(mTemperatureListener);
183     mTemperatureView.setManageBPM(mTemperatureManageDataGraph);
184
185     mRenderThread = new VideoThread(mHolder);
186     mReady = false;
187 }
188
189 /**
190 * Uses the provided {@code width} and {@code height} to measure and layout the inflated
191 * {@link MainView} and {@link BeatingView}.
192 */
193 @Override
194 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
195     // Measure and layout the view with the canvas dimensions.
196     int measuredWidth = View.MeasureSpec.makeMeasureSpec(width, View.MeasureSpec.EXACTLY);
197     int measuredHeight = View.MeasureSpec.makeMeasureSpec(height, View.MeasureSpec.EXACTLY);
198
199     mMainView.measure(measuredWidth, measuredHeight);
200     mMainView.layout(
201         0, 0, mMainView.getMeasuredWidth(), mMainView.getMeasuredHeight());
202
203     mBeatingView.measure(measuredWidth, measuredHeight);
204     mBeatingView.layout(
205         0, 0, mBeatingView.getMeasuredWidth(), mBeatingView.getMeasuredHeight());
206
207     mPhViewer.measure(measuredWidth, measuredHeight);
208     mPhViewer.layout(0, 0, mPhViewer.getMeasuredWidth(), mPhViewer.getMeasuredHeight());
209
210     mTemperatureView.measure(measuredWidth, measuredHeight);
211     mTemperatureView.layout(
212         0, 0, mTemperatureView.getMeasuredWidth(), mTemperatureView.getMeasuredHeight());
213 }
214
215 /**
216 * Keeps the created {@link SurfaceHolder} and updates this class' rendering state.
217 */
218 @Override
```

```
public void surfaceCreated(SurfaceHolder holder) {
    // The creation of a new Surface implicitly resumes the rendering.
    mRenderingPaused = false;
    mHolder = holder;
    updateRenderingState();
}

/**
 * Removes the {@link SurfaceHolder} used for drawing and stops rendering.
 */
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    mHolder = null;
    updateRenderingState();
}

/**
 * Updates this class' rendering state according to the provided {@code paused} flag.
 */
@Override
public void renderingPaused(SurfaceHolder holder, boolean paused) {
    mRenderingPaused = paused;
    updateRenderingState();
}

/**
 * Starts or stops rendering according to the {@link com.google.android.glass.timeline.LiveCard}
 */
private void updateRenderingState() {
    if (mHolder != null && !mRenderingPaused) {
        switch (AppManager.getInstance().getState()) {
            case MENU:
                mRenderThread.quit();
                // mMediaPlayer.setDisplay(null);
                draw(mMainView);
                mBeatingView.stop();
                mPhViewer.stop();
                mTemperatureView.stop();
                mMainView.start();
                break;
            case BEATING:
                // mMediaPlayer.setDisplay(null);
                draw(mBeatingView);
                mMainView.stop();
                mPhViewer.stop();
                mTemperatureView.stop();
                mBeatingView.start();
                break;
        }
    }
}
```

```
269         case PH:
270             // mMediaPlayer.setDisplay(null);
271             draw(mPhViewer);
272             mMainView.stop();
273             mBeatingView.stop();
274             mTemperatureView.stop();
275             mPhViewer.start();
276             break;
277         case TEMPERATURE:
278             // mMediaPlayer.setDisplay(null);
279             draw(mTemperatureView);
280             mMainView.stop();
281             mBeatingView.stop();
282             mPhViewer.stop();
283             mTemperatureView.start();
284             break;
285         case VIDEO:
286             // mMediaPlayer.setDisplay(mHolder);
287             mRenderThread.setShouldRun(true);
288             mRenderThread.start();
289             // mRenderThread.run();
290             mTemperatureView.stop();
291             mBeatingView.stop();
292             mMainView.stop();
293             mPhViewer.stop();
294
295             break;
296         default:
297             // mMediaPlayer.setDisplay(null);
298             draw(mMainView);
299             mPhViewer.stop();
300             mBeatingView.stop();
301             mTemperatureView.stop();
302             mMainView.start();
303             break;
304
305     }
306
307 } else {
308     mMainView.stop();
309     mBeatingView.stop();
310     mPhViewer.stop();
311     mTemperatureView.stop();
312 }
313 }
314
315 /**
316 * Draws the view in the SurfaceHolder's canvas.
317 }
```

```
319     */
320     private void draw(View view) {
321
322         Canvas canvas;
323
324         try {
325             canvas = mHolder.lockCanvas();
326         } catch (Exception e) {
327             Log.e(TAG, "Unable to lock canvas: " + e);
328             return;
329         }
330         if (canvas != null) {
331
332             view.draw(canvas);
333             mHolder.unlockCanvasAndPost(canvas);
334         }
335     }
336
337     /** Setter for the beating graph
338      *
339      * @param bmp , Bitmap which contains the beating graph
340      */
341     public void setBMP(Bitmap bmp){
342
343         this.bmp = bmp;
344         this.mReady = true;
345         Log.i("DRAWER", "bmp settato");
346     }
347
348     public void setSensorGraphs( Map<String,Bitmap> sensorGraphs ){
349         this.mCurrentSensorGraphs = sensorGraphs;
350     }
351
352
353
354     /** Setter for the sensors value hash map
355      *
356      * @param currentSensorValues , Map<String,Double> currentSensorValues which contains the hash
357      *                               map with the current value of the sensors
358      */
359     public void setSensorValues( Map<String,Double> currentSensorValues ){
360         this.mCurrentSensorValues = currentSensorValues;
361     }
362
363
364     public void SetSensorAvg (Map<String,Double> sensorAvg){
365         this.mSensorAverage = sensorAvg;
```

```
367    }  
  
369  
  
371 }
```

Listing A.19: MainView.java

```
1  /*  
2   * Copyright (C) 2013 The Android Open Source Project  
3   *  
4   * Licensed under the Apache License, Version 2.0 (the "License");  
5   * you may not use this file except in compliance with the License.  
6   * You may obtain a copy of the License at  
7   *  
8   *     http://www.apache.org/licenses/LICENSE-2.0  
9   *  
10  * Unless required by applicable law or agreed to in writing, software  
11  * distributed under the License is distributed on an "AS IS" BASIS,  
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
13  * See the License for the specific language governing permissions and  
14  * limitations under the License.  
15  */  
  
17 package com.google.android.glass.sample.klabinterface;  
  
19 import android.content.Context;  
20 import android.media.AudioManager;  
21 import android.media.SoundPool;  
22 import android.os.Handler;  
23 import android.os.SystemClock;  
24 import android.util.AttributeSet;  
25 import android.view.LayoutInflater;  
26 import android.widget.FrameLayout;  
27 import android.widget.TextView;  
  
29 import java.text.SimpleDateFormat;  
30 import java.util.Date;  
31 import java.util.concurrent.TimeUnit;  
  
33 /**  
34  * Animated countdown going from {@code mTimeSeconds} to 0.  
35  *  
36  * The current animation for each second is as follow:  
37  * 1. From 0 to 500ms, move the TextView from {@code MAX_TRANSLATION_Y} to 0 and its alpha from  
38  *     {@code 0} to {@code ALPHA_DELIMITER}.  
39  * 2. From 500ms to 1000ms, update the TextView's alpha from {@code ALPHA_DELIMITER} to {@code 1}.  
40  * At each second change, update the TextView text.  
41  */
```

```
public class MainView extends FrameLayout {  
43  
    /**  
     * Interface to listen for changes in the countdown.  
     */  
47    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;  
57  
  
    private final TextView timeText;  
  
61  
    private final Handler mHandler = new Handler();  
63    private final Runnable mUpdateViewRunnable = new Runnable() {  
  
        @Override  
        public void run() {  
            if (mRunning) {  
                updateView();  
                postDelayed(mUpdateViewRunnable, DELAY_MILLIS);  
            }  
        }  
    };  
73  
  
    private Listener mListener;  
    private boolean mRunning = false;  
77  
    public MainView(Context context) {  
        this(context, null, 0);  
    }  
81  
    public MainView(Context context, AttributeSet attrs) {  
        this(context, attrs, 0);  
    }  
85  
    public MainView(Context context, AttributeSet attrs, int style) {  
        super(context, attrs, style);  
        LayoutInflator.from(context).inflate(R.layout.live_card_layout, this);  
        timeText = (TextView) findViewById(R.id.timestamp);  
    }  
89
```

```
91     }
93
94     /**
95      * Sets a {@link Listener}.
96      */
97     public void setListener(Listener listener) {
98         mListener = listener;
99     }
100
101    /**
102     * Returns the set {@link Listener}.
103     */
104    public Listener getListener() {
105        return mListener;
106    }
107
108    @Override
109    public boolean postDelayed(Runnable action, long delayMillis) {
110        return mHandler.postDelayed(action, delayMillis);
111    }
112
113    /**
114     * Starts the countdown animation if not yet started.
115     */
116    public void start() {
117        if (!mRunning) {
118            postDelayed(mUpdateViewRunnable, 0);
119        }
120        mRunning = true;
121    }
122
123    /**
124     * Stops the chronometer.
125     */
126    public void stop() {
127        if (mRunning) {
128            removeCallbacks(mUpdateViewRunnable);
129            // mStarted = false;
130        }
131        mRunning = false;
132    }
133
134
135    /**
136     * Updates the view to reflect the current state of animation, visible for testing.
137     *
138     * @return whether or not the count down is finished.
139     */
140 }
```

```

    void updateView() {
141    //if (mRunning) {
142        timeText.setText( new SimpleDateFormat("hh:mm a").format(new Date()) );
143        // updateView(millisLeft);
144        if (mListener != null) {
145            mListener.onChange();
146        }
147    }
148}
149
}

```

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 /**
17  * View used to display draw a running Chronometer.
18  *
19  * This code is greatly inspired by the Android's Chronometer widget.
20  */
21
public class BeatingView extends FrameLayout {
22
    boolean mState = false;
23
    private Bitmap bmp;
24
25
    /**
26     * Interface to listen for changes on the view layout.
27     */
28
    public interface Listener {
29
        /** Notified of a change in the view. */
30        public void onChange();
31
    }
32
33
    /**
34     * About 24 FPS, visible for testing.
35     */
36    static final long DELAY_MILLIS = 41;
37
}

```

```
36     private ImageView beatingImage;
38     private final TextView mTitle;
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
60     private Listener mChangeListener;
61
62     private ManageBitmap mManage;
63
64     public BeatingView(Context context) {
65         this(context, null, 0);
66     }
67
68     public BeatingView(Context context, AttributeSet attrs) {
69         this(context, attrs, 0);
70     }
71
72     public BeatingView(Context context, AttributeSet attrs, int style) {
73         super(context, attrs, style);
74         LayoutInflator.from(context).inflate(R.layout.buso_layout, this);
75         mTitle = (TextView) findViewById(R.id.message);
76         beatingImage = (ImageView) findViewById(R.id.image_left);
77         mTitle.setText("Beating");
78         int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/la
79         beatingImage.setImageResource(id);
80
81     }
82
83 }
```

```
86     /**
87      * Sets a {@link Listener}.
88      */
89     public void setListener(Listener listener) {
90         mChangeListener = listener;
91     }
92
93     /**
94      * Returns the set {@link Listener}.
95      */
96     public Listener getListener() {
97         return mChangeListener;
98     }
99
100    /**
101     * Starts the chronometer.
102     */
103    public void start() {
104        if (!mRunning) {
105            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
106        }
107        mRunning = true;
108    }
109
110    /**
111     * Stops the chronometer.
112     */
113    public void stop() {
114        if (mRunning) {
115            removeCallbacks(mUpdateTextRunnable);
116        }
117        mRunning = false;
118    }
119
120    @Override
121    public boolean postDelayed(Runnable action, long delayMillis) {
122        return mHandler.postDelayed(action, delayMillis);
123    }
124
125    @Override
126    public boolean removeCallbacks(Runnable action) {
127        mHandler.removeCallbacks(action);
128        return true;
129    }
130
131    /**
132     * Sets a {@link Listener}.
133     */
134     public void setManageBPM(ManageBitmap manager) {
```

```

134         mManage = manager;
135     }
136
137
138     /**
139      * Updates the value of the chronometer, visible for testing.
140      */
141
142     void updateText() {
143         if (mManage.isReady())
144         {
145             Log.i("viewer", "ce prova");
146             bmp = mManage.getBitmap();
147             beatingImage.setImageBitmap(bmp);
148         }
149         // else
150         //{
151             // int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawa
152             // beatingImage.setImageResource(id);
153         //}
154         if (mChangeListener != null) {
155             mChangeListener.onChange();
156         }
157     }
158
159
160 }
```

Listing A.21: PHViewer.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.LinearLayout;
14 import android.widget.TextView;
15
16 import com.jjoe64.graphview.GraphView;
17 import com.jjoe64.graphview.GraphViewSeries;
18 import com.jjoe64.graphview.LineGraphView;
```

```
19  
21  /**  
22   * View used to display draw a running Chronometer.  
23   *  
24   * This code is greatly inspired by the Android's Chronometer widget.  
25   */  
26  public class PHViewer extends FrameLayout {  
27  
28      private Context context;  
29      private Bitmap bmp;  
30  
31      /**  
32       * Interface to listen for changes on the view layout.  
33       */  
34      public interface Listener {  
35          /** Notified of a change in the view. */  
36          public void onChange();  
37      }  
38  
39      /** About 24 FPS, visible for testing. */  
40      static final long DELAY_MILLIS = 41;  
41  
42      private ImageView beatingImage;  
43      private final TextView mTextTitle;  
44      private TextView AvgView;  
45  
46      private final Handler mHandler = new Handler();  
47      private final Runnable mUpdateTextRunnable = new Runnable() {  
48  
49          @Override  
50          public void run() {  
51              if (mRunning) {  
52                  updateText();  
53                  postDelayed(mUpdateTextRunnable, DELAY_MILLIS);  
54              }  
55          }  
56      };  
57  
58      private boolean mRunning;  
59  
60      public interface ManageBitmap{  
61          public Bitmap getBitmap();  
62  
63          public boolean isReady();  
64  
65          public double getAvg();  
66      }  
67
```

```
69     private Listener mChangeListener;
```

```
71     private ManageBitmap mManage;
```

```
73     public PHViewer(Context context) {
    this(context, null, 0);
}
```

```
75 }
```

```
77     public PHViewer(Context context, AttributeSet attrs) {
    this(context, attrs, 0);
}
```

```
79 }
```

```
81     public PHViewer(Context context, AttributeSet attrs, int style) {
    super(context, attrs, style);
    LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
    beatingImage = (ImageView) findViewById(R.id.image_left);
    mTextTitle = (TextView) findViewById(R.id.message);
    AvgView = (TextView) findViewById(R.id.avg);
    mTextTitle.setText("PH");
    int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/la
    beatingImage.setImageResource(id);
//        int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/
//        beatingImage.setImageResource(id);

93 }
95 
```

```
97 /**
 * Sets a {@link Listener}.
 */
99 public void setListener(Listener listener) {
    mChangeListener = listener;
}

103 /**
 * Returns the set {@link Listener}.
 */
105 public Listener getListener() {
    return mChangeListener;
}

109 }

111 /**
 * Starts the chronometer.
 */
113 public void start() {
    if (!mRunning) {
        postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
    }
}
```

```
117         }
118         mRunning = true;
119     }

121     /**
122      * Stops the chronometer.
123     */
124     public void stop() {
125         if (mRunning) {
126             removeCallbacks(mUpdateTextRunnable);
127         }
128         mRunning = false;
129     }

131     @Override
132     public boolean postDelayed(Runnable action, long delayMillis) {
133         return mHandler.postDelayed(action, delayMillis);
134     }

136     @Override
137     public boolean removeCallbacks(Runnable action) {
138         mHandler.removeCallbacks(action);
139         return true;
140     }

142     /**
143      * Sets a {@link Listener}.
144     */
145     public void setManageBPM(ManageBitmap manager) {
146         mManage = manager;
147     }

149     /**
150      * Updates the value of the chronometer, visible for testing.
151     */
152     void updateText() {
153         if (mManage.isReady()) {
154             Log.i("PH", "ce prova");
155             bmp = mManage.getBitmap();
156             AvgView.setText(Double.toString(mManage.getAvg()));
157             beatingImage.setImageBitmap(bmp);

159

161         }
162         // else
163         //{
164             // int id = getResources().getIdentifier("com.google.android.sample.stopwatch:drawable/
165             // beatingImage.setImageResource(id);
```

```
    //}
167    if (mChangeListener != null) {
168        mChangeListener.onChange();
169    }
170}
171
172
173
174
175}
```

Listing A.22: TemperatureView.java

```
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 * View used to display draw a running Chronometer.
18 *
19 * This code is greatly inspired by the Android's Chronometer widget.
20 */
21
22 public class TemperatureView extends FrameLayout {
23
24     private Bitmap bmp;
25     private TextView AvgView;
26
27 /**
28 * Interface to listen for changes on the view layout.
29 */
30
31     public interface Listener {
32         /** Notified of a change in the view. */
33         public void onChange();
34     }
35
36     /** About 24 FPS, visible for testing. */
37     static final long DELAY_MILLIS = 41;
```

```
38     private ImageView beatingImage;
39     private final TextView mTitle;
40
41     private final Handler mHandler = new Handler();
42     private final Runnable mUpdateTextRunnable = new Runnable() {
43
44         @Override
45         public void run() {
46             if (mRunning) {
47                 updateText();
48                 postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
49             }
50         }
51     };
52
53     private boolean mRunning;
54
55     public interface ManageBitmap{
56         public Bitmap getBitmap();
57
58         public boolean isReady();
59
60         public double getAvg();
61     }
62
63     private Listener mChangeListener;
64
65     private ManageBitmap mManage;
66
67     public TemperatureView(Context context) {
68         this(context, null, 0);
69     }
70
71     public TemperatureView(Context context, AttributeSet attrs) {
72         this(context, attrs, 0);
73     }
74
75     public TemperatureView(Context context, AttributeSet attrs, int style) {
76         super(context, attrs, style);
77         LayoutInflator.from(context).inflate(R.layout.buso_layout, this);
78         beatingImage = (ImageView) findViewById(R.id.image_left);
79         mTitle = (TextView) findViewById(R.id.message);
80         AvgView = (TextView) findViewById(R.id.avg);
81
82         mTitle.setText("Temperature");
83         int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/lay
84         beatingImage.setImageResource(id);
```

```
86
87    }
88
89
90    /**
91     * Sets a {@link Listener}.
92     */
93
94    public void setListener(Listener listener) {
95        mChangeListener = listener;
96    }
97
98    /**
99     * Returns the set {@link Listener}.
100    */
101
102    public Listener getListener() {
103        return mChangeListener;
104    }
105
106    /**
107     * Starts the chronometer.
108     */
109
110    public void start() {
111        if (!mRunning) {
112            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
113        }
114        mRunning = true;
115    }
116
117    /**
118     * Stops the chronometer.
119     */
120
121    public void stop() {
122        if (mRunning) {
123            removeCallbacks(mUpdateTextRunnable);
124        }
125        mRunning = false;
126    }
127
128    @Override
129    public boolean postDelayed(Runnable action, long delayMillis) {
130        return mHandler.postDelayed(action, delayMillis);
131    }
132
133    @Override
134    public boolean removeCallbacks(Runnable action) {
135        mHandler.removeCallbacks(action);
136        return true;
137    }
138
139    /**
140     *
```

```
* Sets a {@link Listener}.
```

```
 */
public void setManageBPM(ManageBitmap manager) {
    mManage = manager;
}

/**
 * Updates the value of the chronometer, visible for testing.
 */
void updateText() {
    if (mManage.isReady())
    {
        Log.i("Temperature", "ce prova");
        bmp = mManage.getBitmap();
        AvgView.setText(Double.toString(mManage.getAvg()));
        beatingImage.setImageBitmap(bmp);
    }
    // else
    //{
    //    int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/beatingImage");
    //    beatingImage.setImageResource(id);
    //}
    if (mChangeListener != null) {
        mChangeListener.onChange();
    }
}
}

}
```

Listing A.23: VideoPlayerActivity.java

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

3 import android.app.Activity;
import android.content.Intent;
5 import android.content.res.AssetManager;
import android.net.Uri;
7 import android.os.Bundle;
import android.os.Environment;
9 import android.util.Log;
import android.view.Menu;
11 import android.view.MenuItem;
import android.widget.MediaController;
13 import android.widget.VideoView;
```

```
15 import java.io.File;
16 import java.io.FileOutputStream;
17 import java.io.IOException;
18 import java.io.InputStream;
19 import java.io.OutputStream;
20
21 public class VideoPlayerActivity extends Activity {
22     private static final int VIDEO_PLAY_REQUEST_CODE = 200;
23     private static final String TAG = "VIDEO_TAG";
24     public void onCreate(Bundle savedInstanceState) {
25         super.onCreate(savedInstanceState);
26         String filepath;
27         Bundle extras = getIntent().getExtras();
28         if (extras != null)
29             filepath = extras.getString("filepath");
30         else {
31             filepath = copyAsset(VIDEO_FILE_NAME);
32         }
33
34         Intent i = new Intent();
35         i.setAction("com.google.glass.action.VIDEOPLAYER");
36         i.putExtra("video_url", filepath);
37         startActivityForResult(i, VIDEO_PLAY_REQUEST_CODE);
38     }
39     protected void onActivityResult(int requestCode, int resultCode, Intent data) {
40         if (requestCode == VIDEO_PLAY_REQUEST_CODE )
41             finish();
42     }
43     String copyAsset(String filename) {
44         final String PATH = Environment.getExternalStorageDirectory().toString() + "/myvideoapps/";
45         File dir = new File(PATH);
46         if (!dir.exists()) {
47             if (!dir.mkdirs()) {
48                 Log.v(TAG, "ERROR: Creation of directory " + PATH + " on sdcard failed");
49                 return null;
50             } else {
51                 Log.v(TAG, "Created directory " + PATH + " on sdcard");
52             }
53         }
54         if (!(new File( PATH + filename).exists())) {
55             try {
56                 AssetManager assetManager = getAssets();
57                 InputStream in = assetManager.open(filename);
58                 OutputStream out = new FileOutputStream(PATH + filename);
59                 byte[] buf = new byte[1024];
60                 int len;
61                 while ((len = in.read(buf)) > 0) {
62                     out.write(buf, 0, len);
63                 }
64             }
65         }
66     }
67 }
```

```

63         in.close();
64         out.close();
65     } catch (IOException e) {
66         Log.e(TAG, "Was unable to copy " + filename + e.toString());
67         return null;
68     }
69 }
70 return PATH + filename;
71 }
72 }
```

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
14 /**
15 * Activity showing the stopwatch options menu.
16 */
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
31
32     private final Handler mHandler = new Handler();
33     private int state = 0;
34
35     @Override
36     public void onAttachedToWindow() {
37         super.onAttachedToWindow();
38     }
39 }
```

```
    openOptionsMenu();
38 }

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

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

49         boolean initialView = appManager.getState() == MENU;
50         boolean imageView = appManager.getState() == PH ||
51             appManager.getState() == TEMPERATURE ||
52             appManager.getState() == BEATING ||
53             appManager.getState() == VIDEO;

54         setOptionsMenuState(menu, R.id.action_back, imageView);
55         setOptionsMenuState(menu, R.id.action_view_ph, initialView);
56         setOptionsMenuState(menu, R.id.action_view_temperature, initialView);
57         setOptionsMenuState(menu, R.id.action_view_video, initialView);
58         setOptionsMenuState(menu, R.id.action_view_beating, initialView);
59         setOptionsMenuState(menu, R.id.action_stop, true);

60         return true;
61     }

62     @Override
63     public boolean onOptionsItemSelected(MenuItem item) {
64         // Handle item selection.
65         switch (item.getItemId()) {
66             case R.id.action_stop:
67                 // Stop the service at the end of the message queue for proper options menu
68                 // animation. This is only needed when starting a new Activity or stopping a Service
69                 // that published a LiveCard.
70                 post(new Runnable() {
71
72                     @Override
73                     public void run() {
74                         stopService(new Intent(MenuActivity.this, StopwatchService.class));
75                     }
76                 });
77                 return true;
78             case R.id.action_view_beating:
79                 handleViewBeating();
80                 return true;
81         }
82     }
83 }
```

```
86         case R.id.action_back:
87             handleViewBack();
88             return true;
89         case R.id.action_view_temperature:
90             handleViewTemperature();
91             return true;
92         case R.id.action_view_ph:
93             handleViewPh();
94             return true;
95         case R.id.action_view_video:
96             handleViewVideo();
97             return true;
98         default:
99             return super.onOptionsItemSelected(item);
100     }
101 }
102
103 private void handleViewVideo() {
104     Toast.makeText(this, "Video", Toast.LENGTH_SHORT).show();
105     // launch a new thread for starting a new live card
106     mHandler.post(new Runnable() {
107         @Override
108         public void run() {
109             Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
110             intent.putExtra(Intent.EXTRA_TEXT, "Video");
111             startService(intent);
112         }
113     });
114 }
115
116 private void handleViewTemperature() {
117     Toast.makeText(this, "Temperature", Toast.LENGTH_SHORT).show();
118     // launch a new thread for starting a new live card
119     mHandler.post(new Runnable() {
120         @Override
121         public void run() {
122             Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
123             intent.putExtra(Intent.EXTRA_TEXT, "Temperature");
124             startService(intent);
125         }
126     });
127 }
128
129 private void handleViewPh() {
130     Toast.makeText(this, "Ph", Toast.LENGTH_SHORT).show();
131     // launch a new thread for starting a new live card
132     mHandler.post(new Runnable() {
133         @Override
134         public void run() {
```

```
136         Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
137         intent.putExtra(Intent.EXTRA_TEXT, "pH");
138         startService(intent);
139     }
140 }
141
142 private void handleViewBack() {
143     Toast.makeText(this, "Menu", 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, StopwatchService.class);
149             intent.putExtra(Intent.EXTRA_TEXT, "Menu");
150             startService(intent);
151         }
152     });
153 }
154
155 @Override
156 public void onOptionsMenuClosed(Menu menu) {
157     // Nothing else to do, closing the Activity.
158     finish();
159 }
160
161 /**
162 * Posts a {@link Runnable} at the end of the message loop, overridable for testing.
163 */
164 protected void post(Runnable runnable) {
165     mHandler.post(runnable);
166 }
167
168 /**
169 * The function handle the request to view the beating plot
170 */
171 private void handleViewBeating() {
172     Toast.makeText(this, "Beating", Toast.LENGTH_SHORT).show();
173     // launch a new thread for starting a new live card
174     mHandler.post(new Runnable() {
175         @Override
176         public void run() {
177             Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
178             intent.putExtra(Intent.EXTRA_TEXT, "Beating");
179             startService(intent);
180         }
181     });
182 }
```

```

184     private static void setOptionsMenuState(Menu menu, int menuItemId, boolean enabled){
185         MenuItem menuItem = menu.findItem(menuItemId);
186         menuItem.setVisible(enabled);
187         menuItem.setEnabled(enabled);
188     }
189 }

```

Listing A.25: AppManager.java

```

package com.google.android.glass.sample.klabinterface;
2
3 /**
4 * Created by alik on 12/2/2014.
5 */
6 public class AppManager {
7
8     private int state;
9
10    private static AppManager instance = new AppManager();
11
12    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
24 }

```

Listing A.26: DataPoint.java

```

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     }

```

```
14     public double getTimestamp() {  
15         return mTimestamp;  
16     }  
17     public double getValue() {  
18         return mValue;  
19     }  
20 }
```

[7]

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	Driver for LED	49
Figure 43	LEDs experiments board	50
Figure 44	Circuit mounted on breadboard top view	51
Figure 45	Circuit mounted on breadboard side view	52

LIST OF LISTINGS

A.1	KlabFirware (bash)	53
A.2	SensorAcquiring (bash)	54
A.3	Pins Setting	54
A.4	ElectrovalvesDriver.cpp	55
A.5	sensor_aquiring.cpp	57
A.6	compute_beating.cpp	60
A.7	Pins Setting	61
A.8	HTTP.h	62
A.9	HTTP.cpp	63
A.10	main.cpp	64
A.11	VideoStoring.pro	64
A.12	mytimer.h	65
A.13	mytimer.cpp	65
A.14	downloader.h	66
A.15	downloader.cpp	67
A.16	Main Script of Server	69
A.17	MainService.java	76
A.18	AppDrawer.java	91
A.19	MainView.java	99
A.20	BeatingView.java	102
A.21	PHViewer.java	105
A.22	TemperatureView.java	109
A.23	VideoPlayerActivity.java	112
A.24	MenuActivity.java	114
A.25	AppManager.java	118
A.26	DataPoint.java	118

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] 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>.