POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

Tesi di Laurea Magistrale

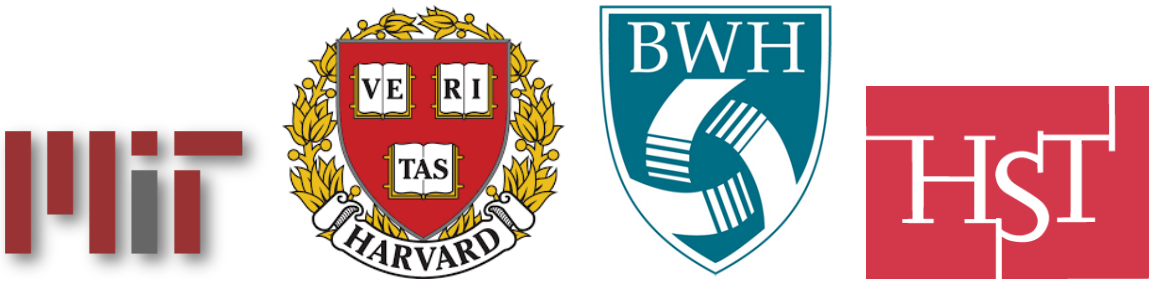
Google Glass Data Visualization and Monitoring for Organs-on-a-Chip and Biomedical Applications

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* (Cambdridge, MA, USA), Harvard-MIT Health Science and Technology, Brigham and Women's Hospital under the supervision of professor Ali Khademhosseini and Ph.D. Yu Shrike Zhang.
The design which is going to be described, has been inserted inside the context of a five years project (***XCEL*** grant), sponsored by the U.S. Defense Threat Reduction Agency (*DTRA*).

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

This master thesis will focus on designing a custom user interface on *Google Glass* for simultaneous recording of biosensing data such as temperature, pH, and microscopy images/videos as well as remote control of microfluidic valves and devices. The project involves all the hierarchical layers, starting from the physical one with the circuit in charge to acquire data from bio-sensors and drive the valves, up to the glasswear[1].

In the Introduction chapter, the main keys of the project are presented in detail as well as the final result from a user point of view.
After that, a detailed description of each abstraction level which goes to build the entire systems is shown: starting from the bottom (Hardware) reaching the top (Google Glass Application) passing through the Firmware, that runs in an embedded *Linux* platform, and the Software, present on the *PC* and the *Google App Engine*.
The Experiments and Conclusion chapter shows the obtained results with different experiments. The thesis ends discussing the limitations and improvements of the system.

[1] Google Glass Application

CONTENTS

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 containg four 3D fully-functional *organs-on-a-chip*.

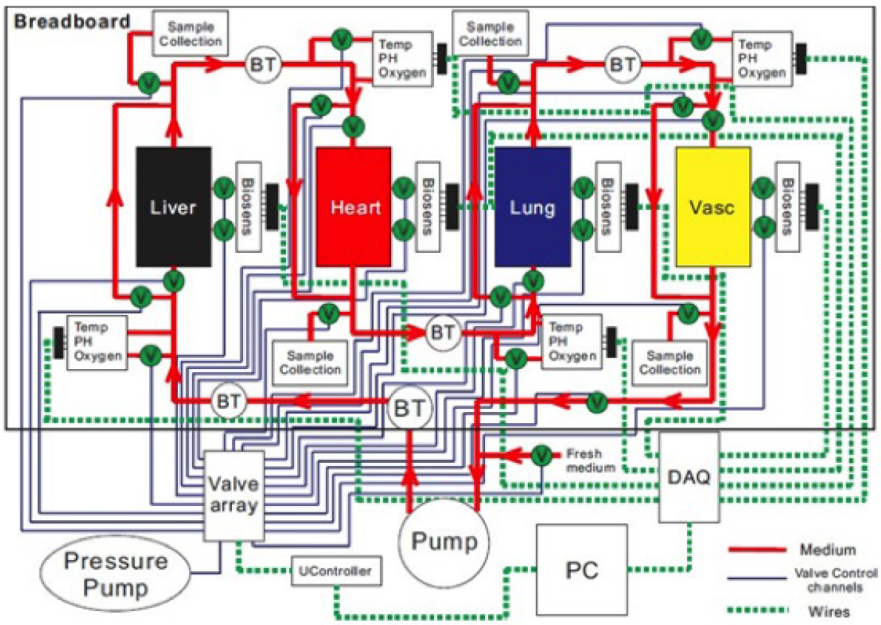


Figure 1: *XCEL* project (*Body-on-a-chip*)

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

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

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

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

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

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

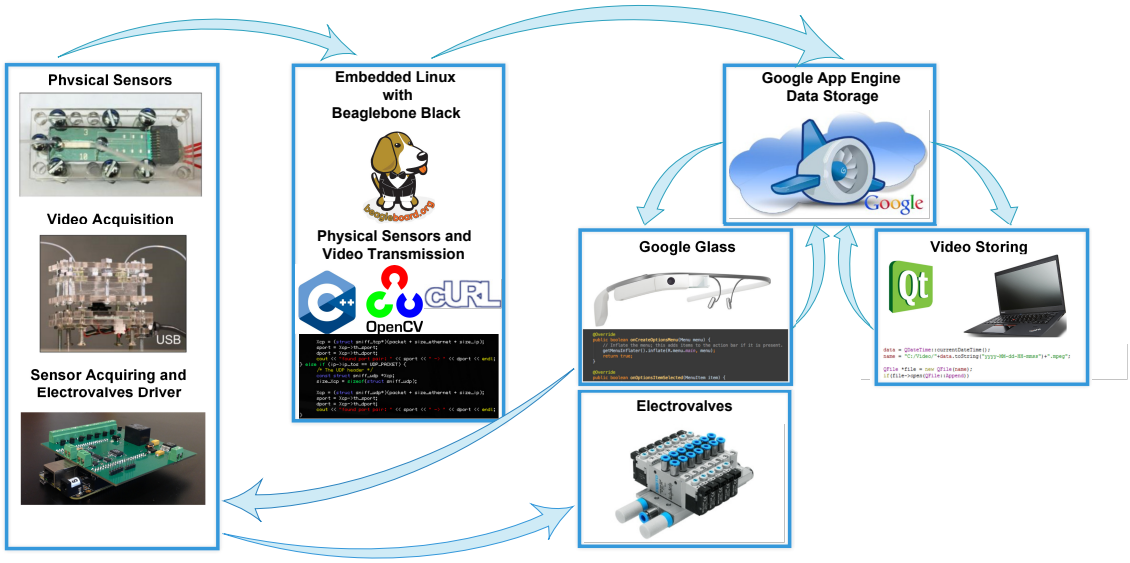


Figure 2: Block diagram of the system

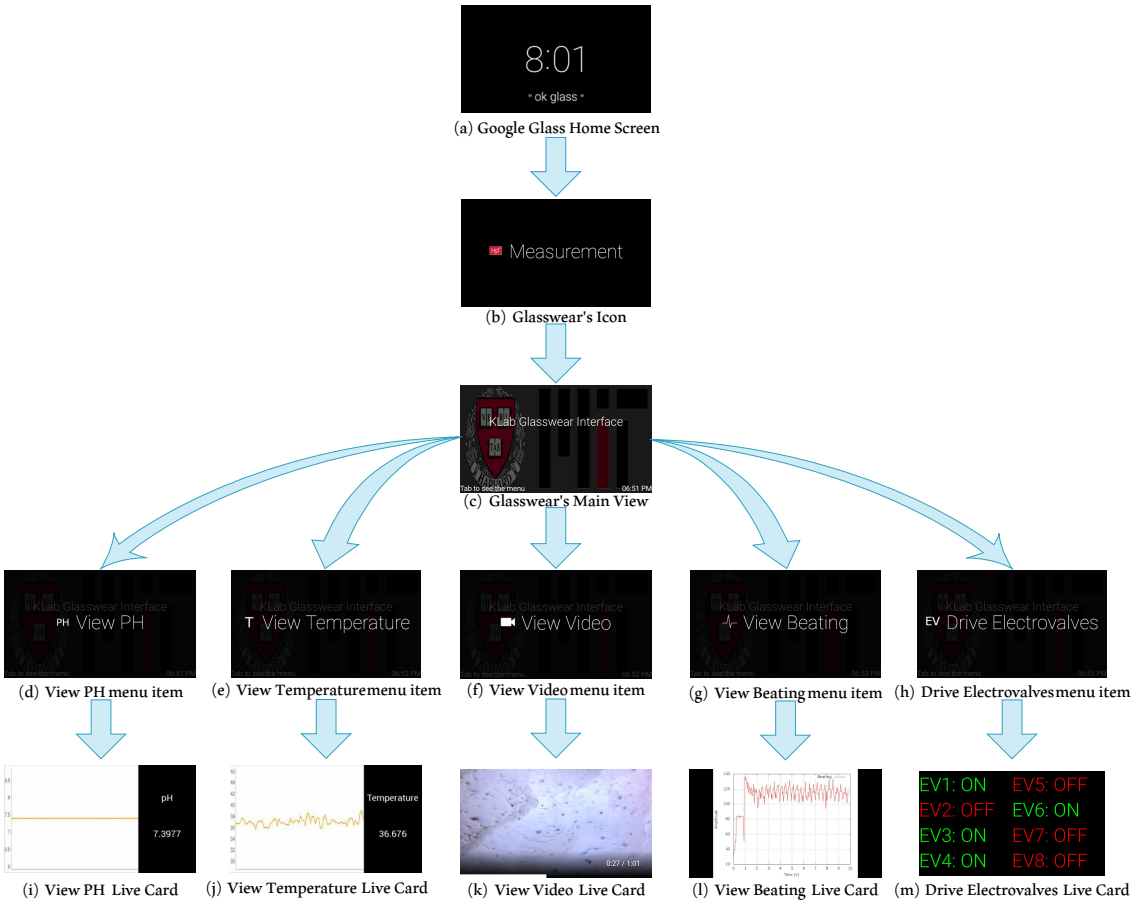
The (Fig.2) shows the principal steps of data transmission from physical and video sensors to the Google Glass via an *Embedded Linux System* performed using the **Beaglebone Black**.
The Beaglebone Black runs processes that are in charged to:

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

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

THE GLASSWEAR

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



(a) Google Glass Home Screen
(b) Glasswear's Icon
(c) Glasswear's Main View
(d) View PH menu item
(e) View Temperature menu item
(f) View Video menu item
(g) View Beating menu item
(h) Drive Electrovalves menu item
(i) View PH Live Card
(j) View Temperature Live Card
(k) View Video Live Card
(l) View Beating Live Card
(m) Drive Electrovalves Live Card

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).

(a) Elecrovalves Status View (b) Toggle EV1 menu item (c) Toggled EV1

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

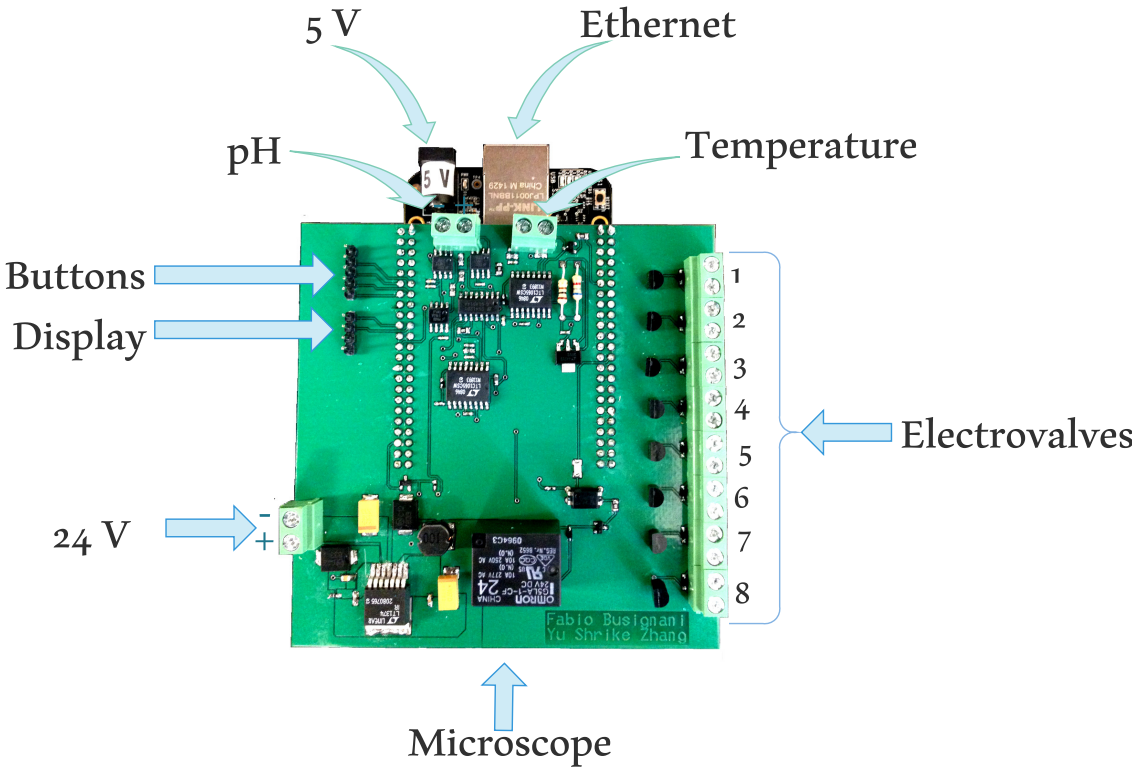


Figure 7: The Board

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 Baglebone Black and by the conditiong 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 sensor[2];
- *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 required the interaction with Google Glass, such as sensors calibration.

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

VIDEO STORING

The storing of the microscope video plays an important role of this system. It may be essential to review the recorded video during the experiment and in order to fulfill this aim a *Qt* program has been designed.
I chose *Qt* because in this way the program is available for different operating systems, *Linux*, *Windows*, and *MacOS*.

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

Figure 8: Storing Microscope Video Program's Icon

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

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

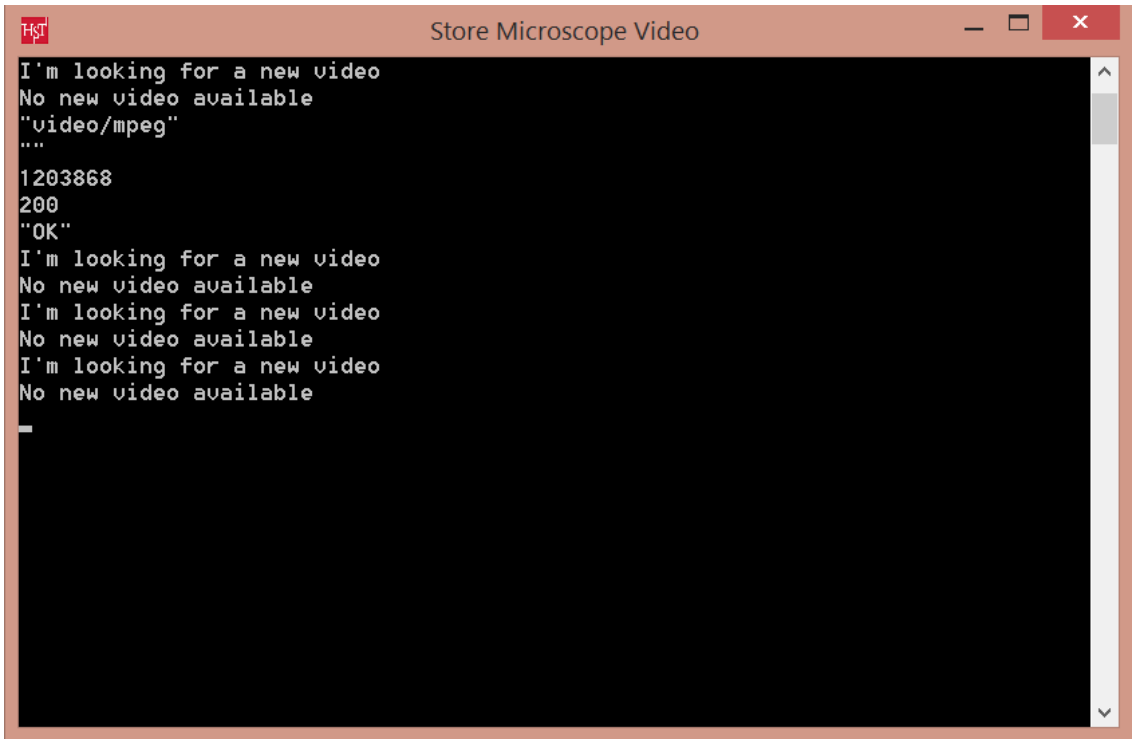


Figure 9: Storing Microscope Video Console

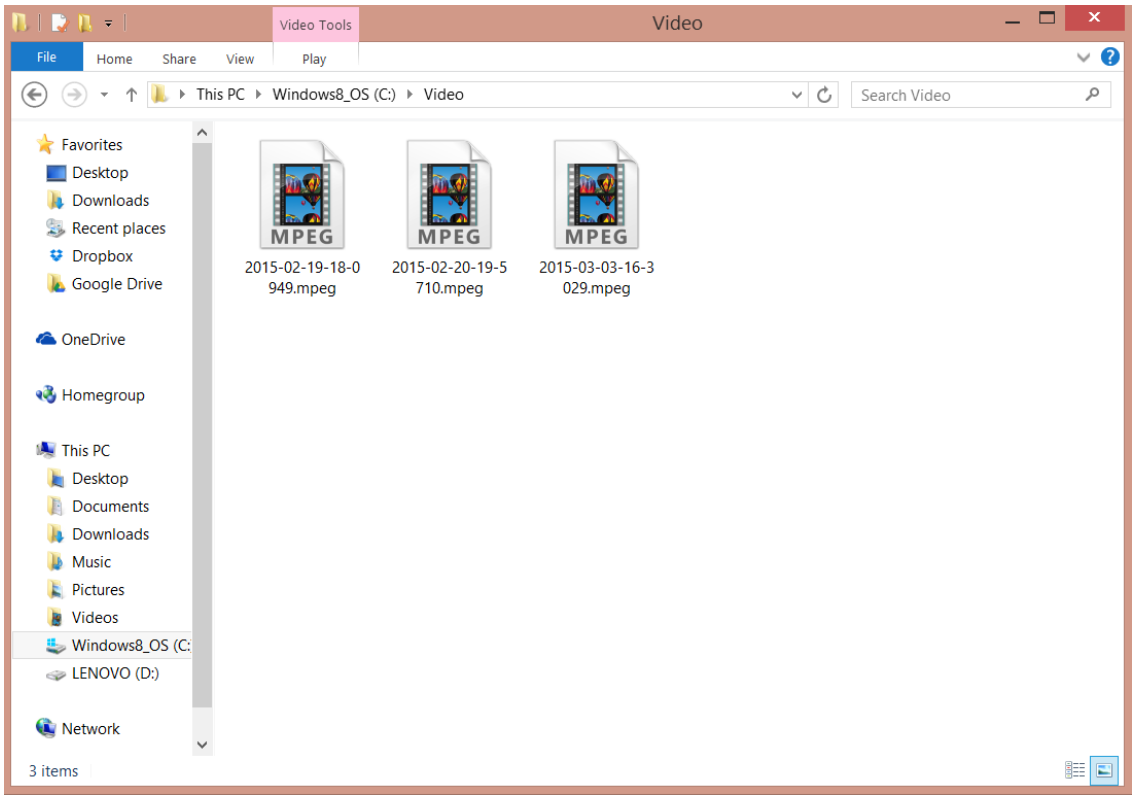


Figure 10: Video Stored in the Folder

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.

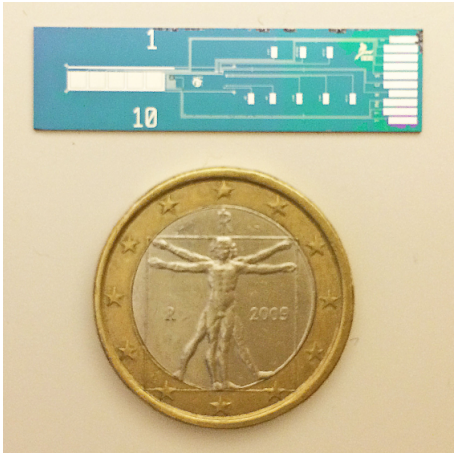


Figure 11: Multisensor - Size Comparison

The (Fig.11) shows the sensor and highlights its very small dimension ($35.1\ x\ 9.3\ mm$). 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.

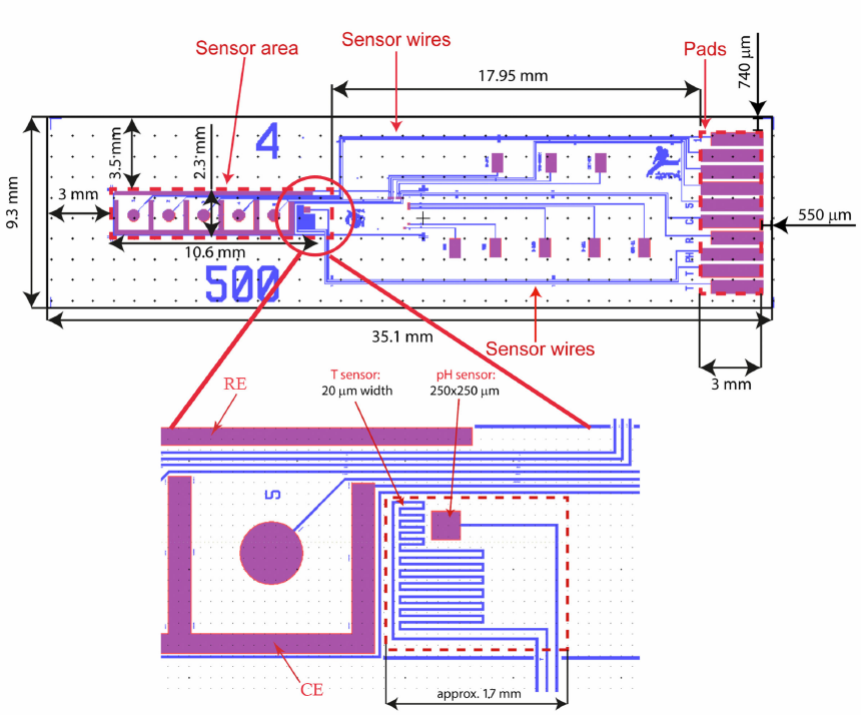


Figure 12: Multisensor - Scheme

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 $\pm 0.001\ ^{\circ}C$) 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).

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

Where:

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

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

$$A = \alpha + \frac{\alpha \cdot \delta}{100}, \tag{2a}$$

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

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

And $\alpha = 0.003921 \Omega/°C$, $\beta = 0$ for positive temperature and $\delta = 1.49$.

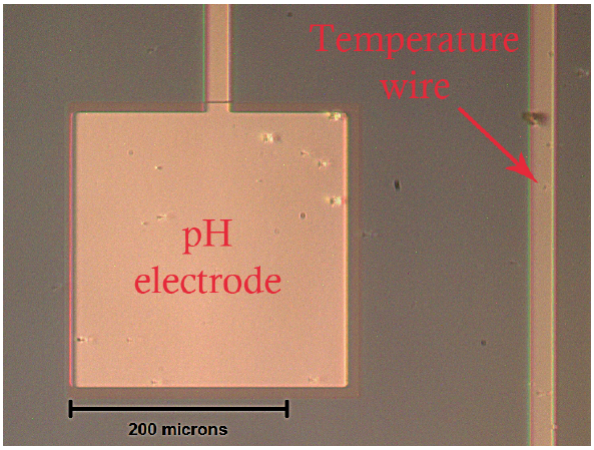


Figure 13: PH and Temperature Sensors from an Optical Image

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.

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

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

- $pH\,(X)$, is the pH value of unknown solution;
- $pH\,(S)$, 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 ($9.6485309 \cdot 10^4\ C\ mol^{-1}$);
- R, is the universal gas constant ($8.314510\ J\ K^{-1}\ mol^{-1}$);
- 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).

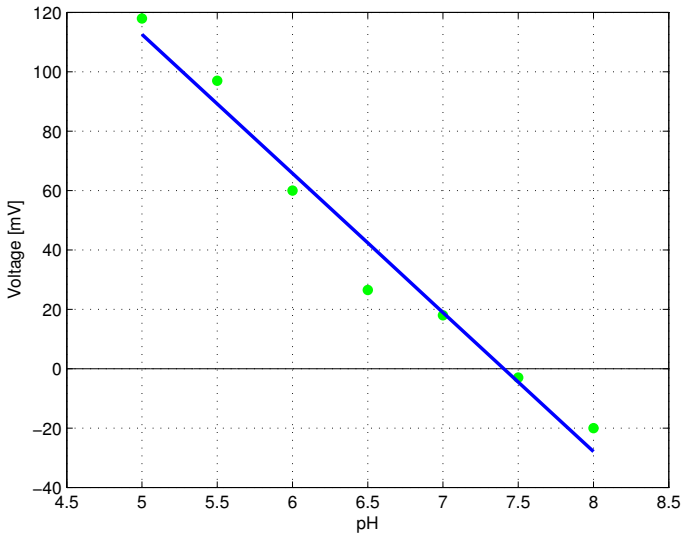


Figure 14: Typical pH-sensor transfer function

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.
Fist 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 $10\ K\Omega$ 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.

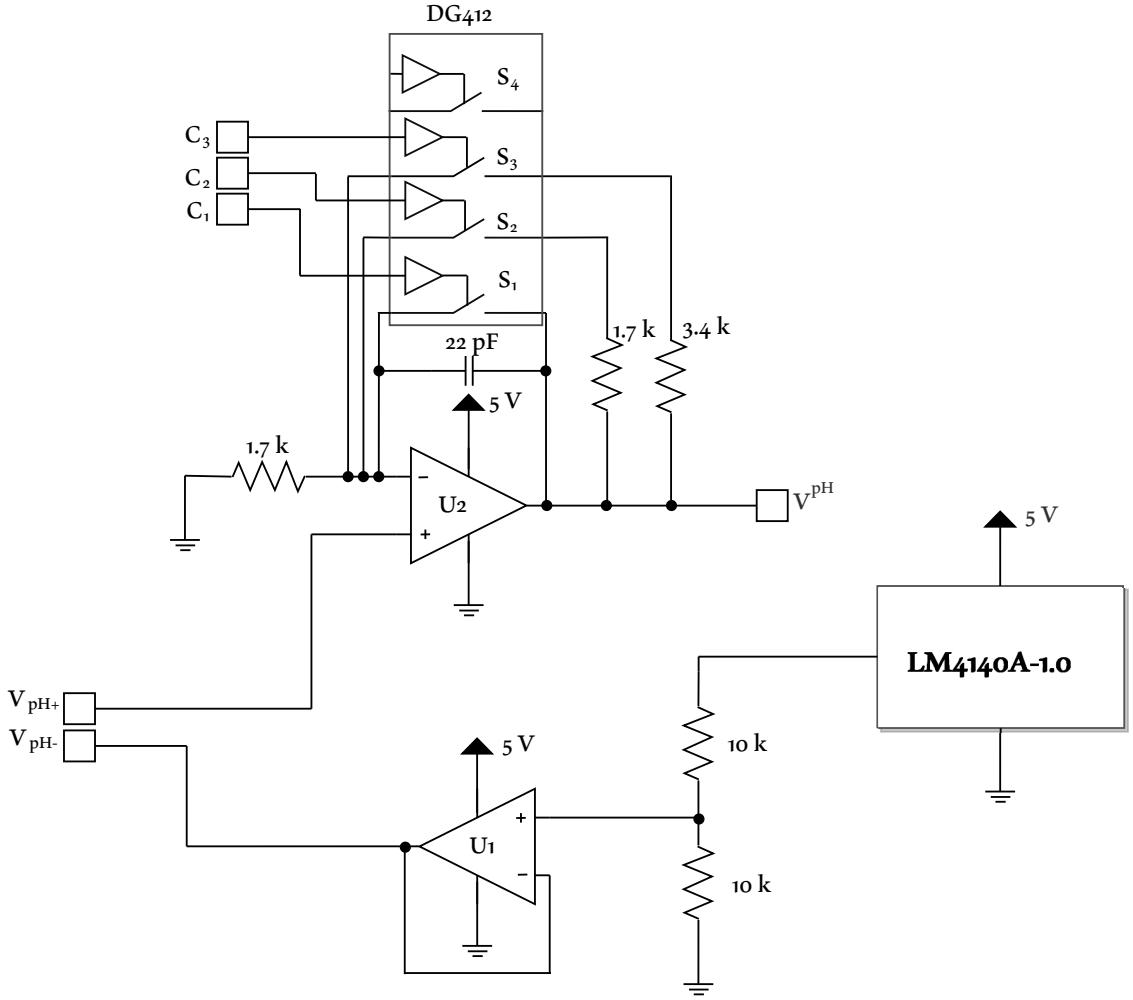


Figure 15: Conditioning circuit for pH sensor

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:

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

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 ($3 \pm 17\ fA$).

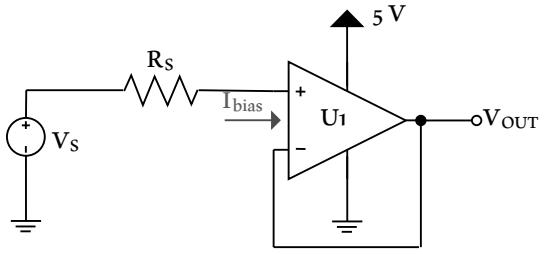


Figure 16: Error caused by Amplifier's Input Bias Current

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

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.

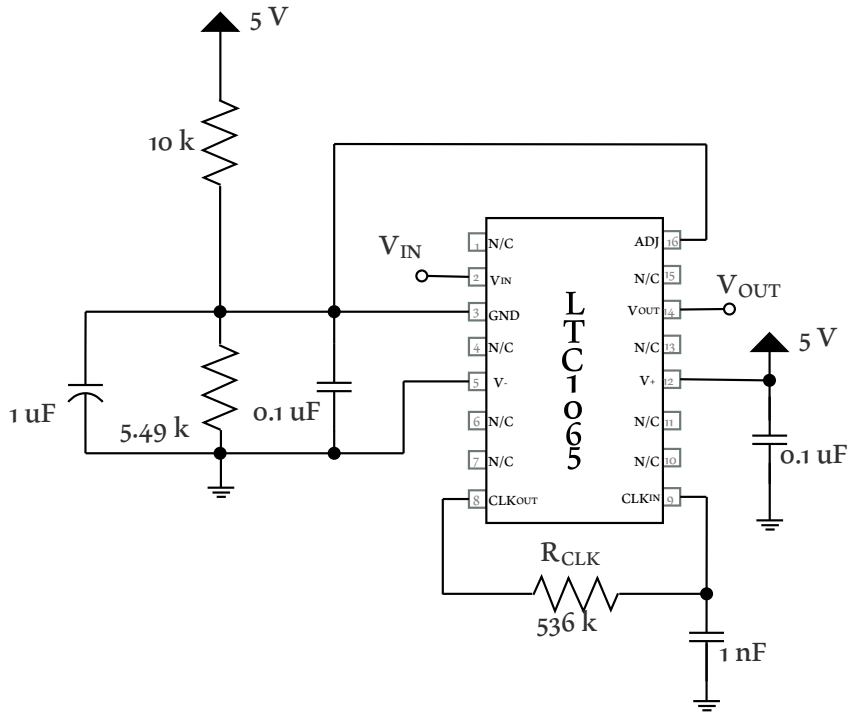


Figure 17: Low-Pass Filter Schematic

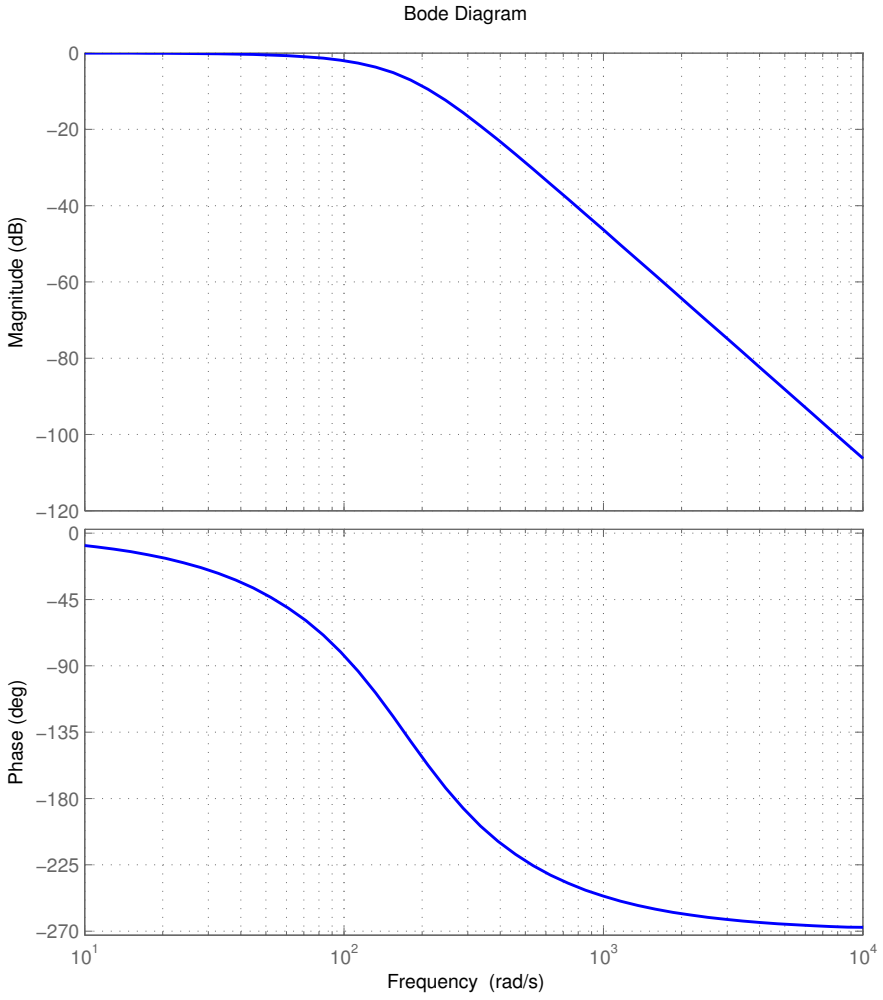


Figure 18: Low-Pass Filter Frequency Response

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

Connecting together al 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.

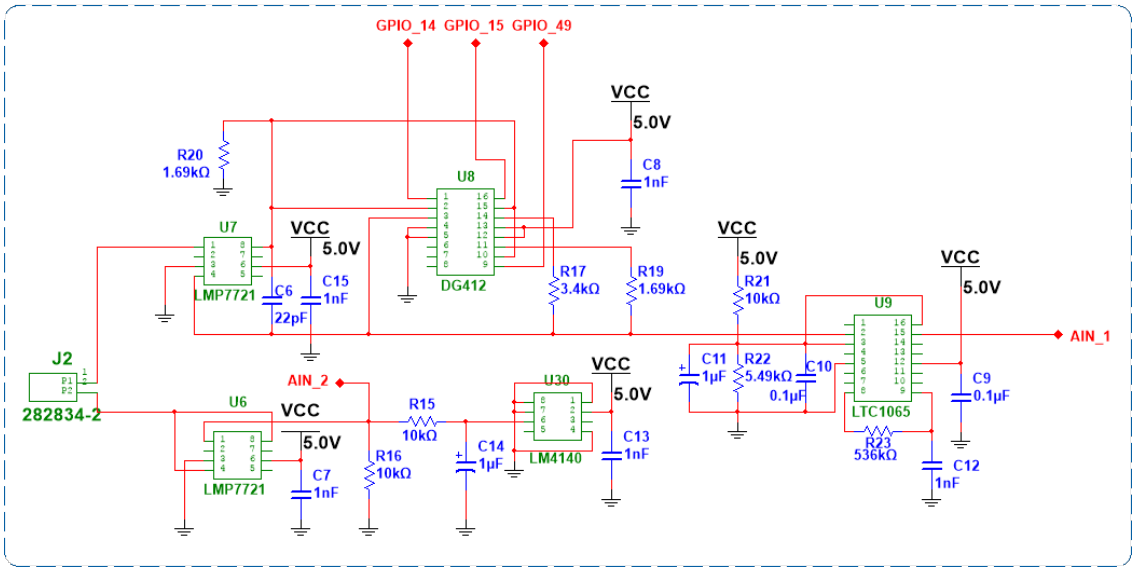


Figure 19: Acquisition Path for pH Sensor

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:

$$V_T = 680\mu \cdot R_T \tag{5}$$

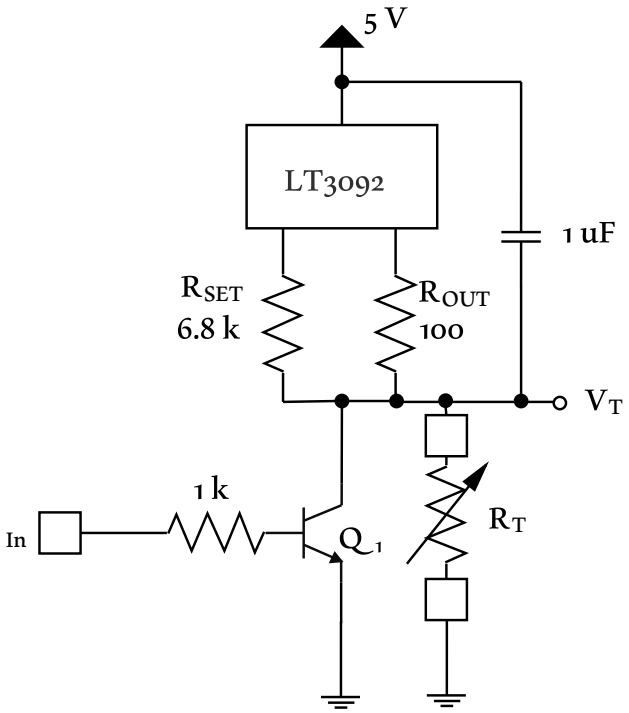


Figure 20: Conditioning Circuit for Temperature Sensor

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

$$I_{OUT} = 10\mu \cdot \frac{R_{SET}}{R_{OUT}} \tag{6}$$

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 on.

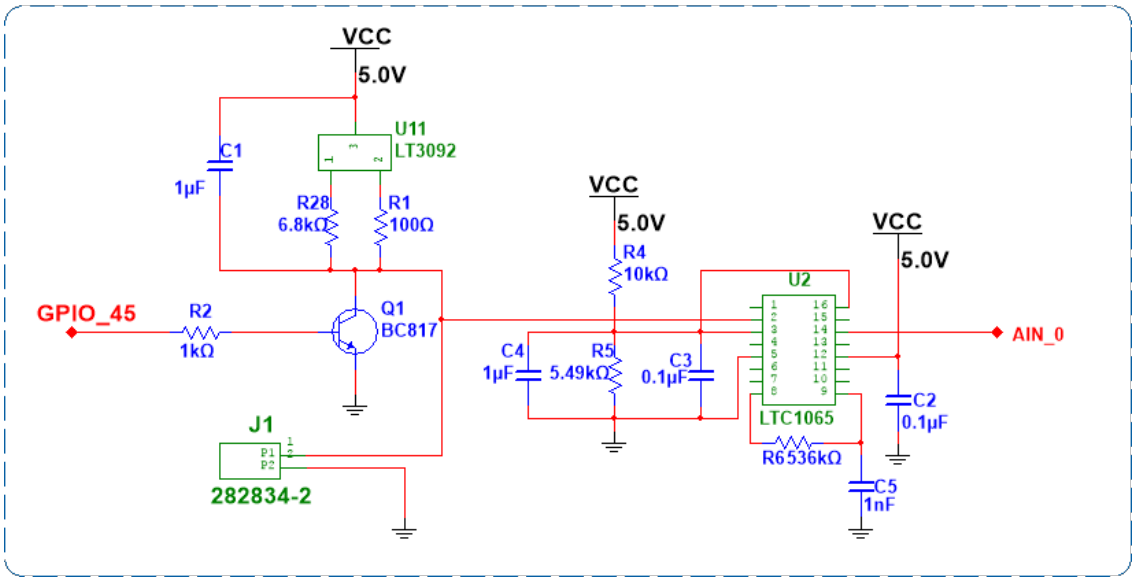


Figure 21: Acquisition Path for pH Sensor

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-Chnnel 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 1N4148 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

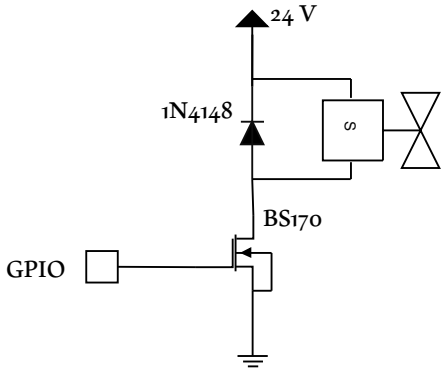


Figure 23: Driver for electrovalves

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.

Trough 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.

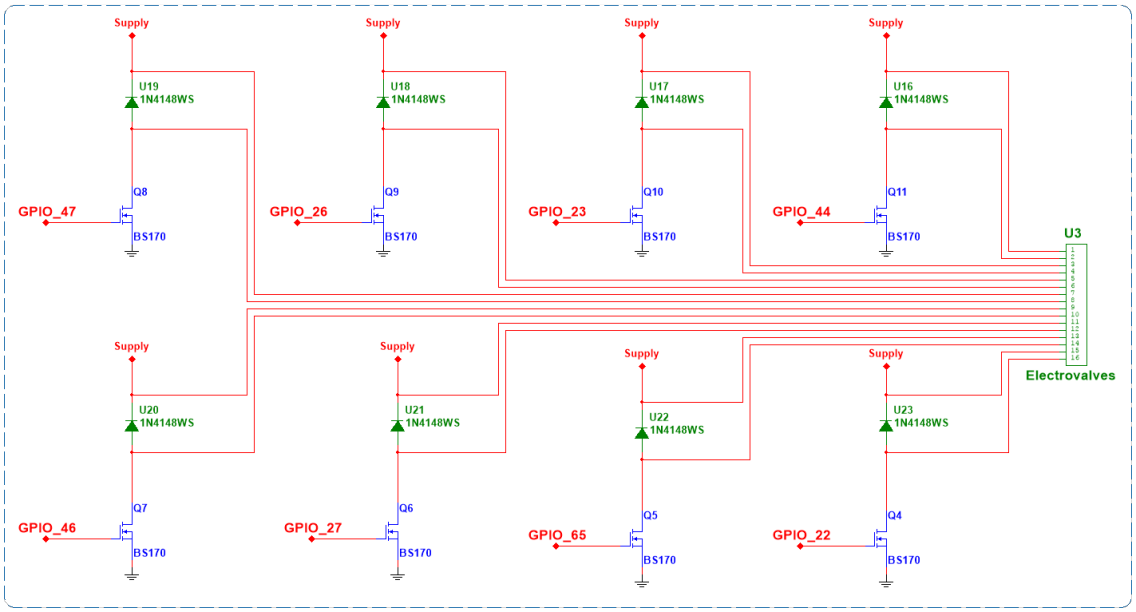


Figure 24: Electrovalves Driver Circuit

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:

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

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

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.

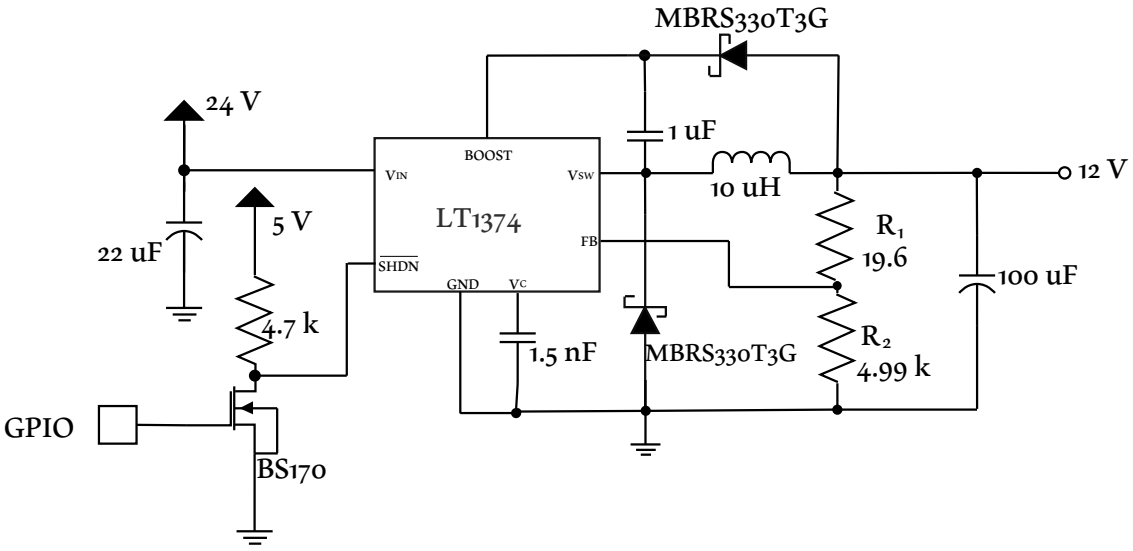


Figure 25: DC-DC circuit

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

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

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 $0.05\ \Omega$ to $0.2\ \Omega$, 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).

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

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 $1.5\ nF$.

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

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 Beglebone 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.

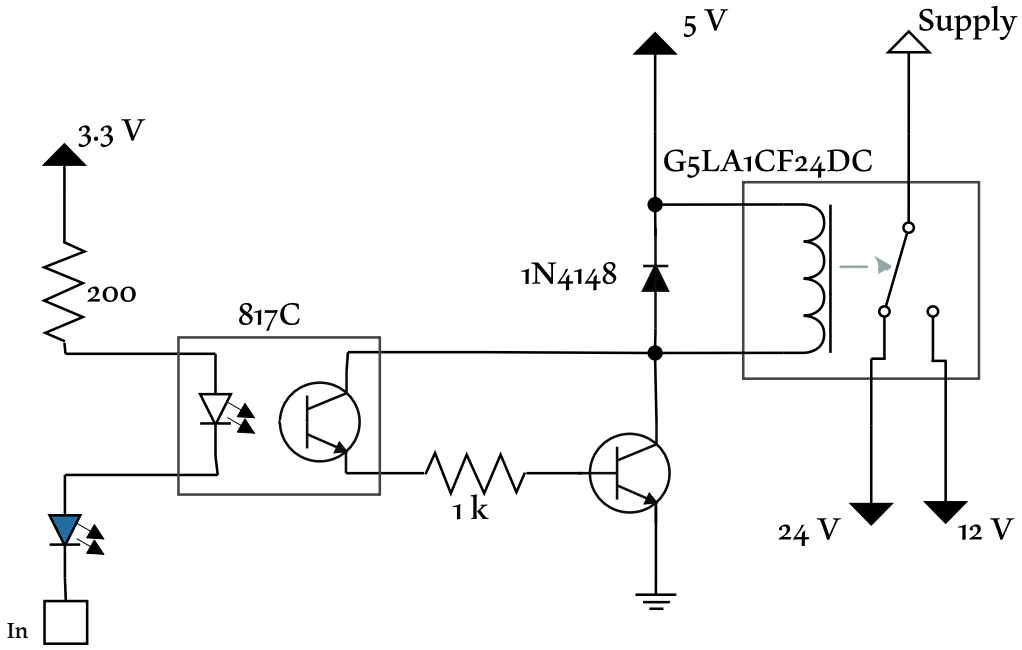


Figure 26: Relay circuit

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 $12\ V$.

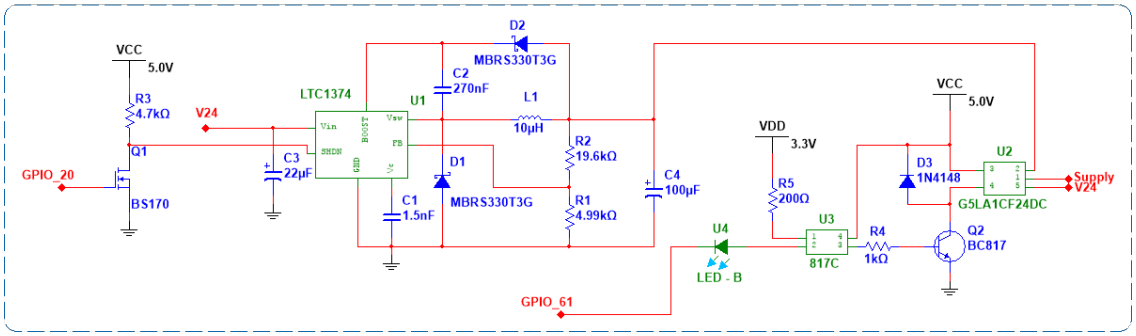


Figure 27: Electrovalves Supply Circuit

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

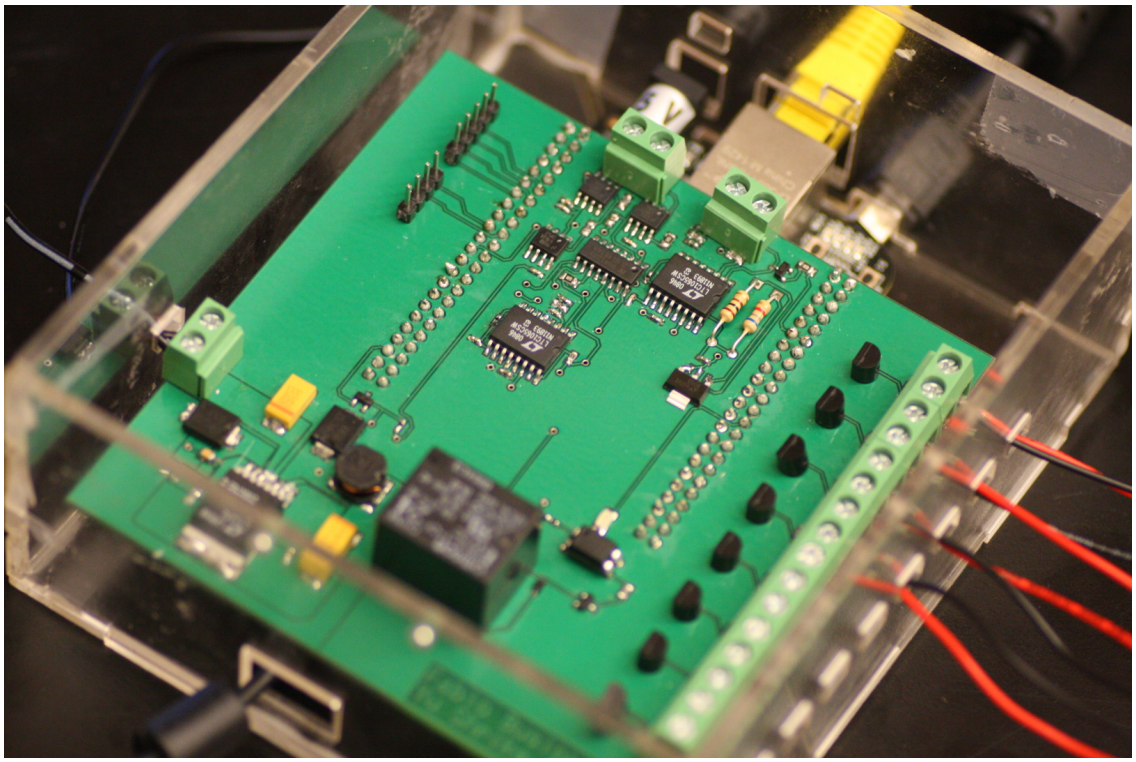


Figure 28: Final system mounted on a PCB

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 ($100\ x\ 100\ mm$) 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.

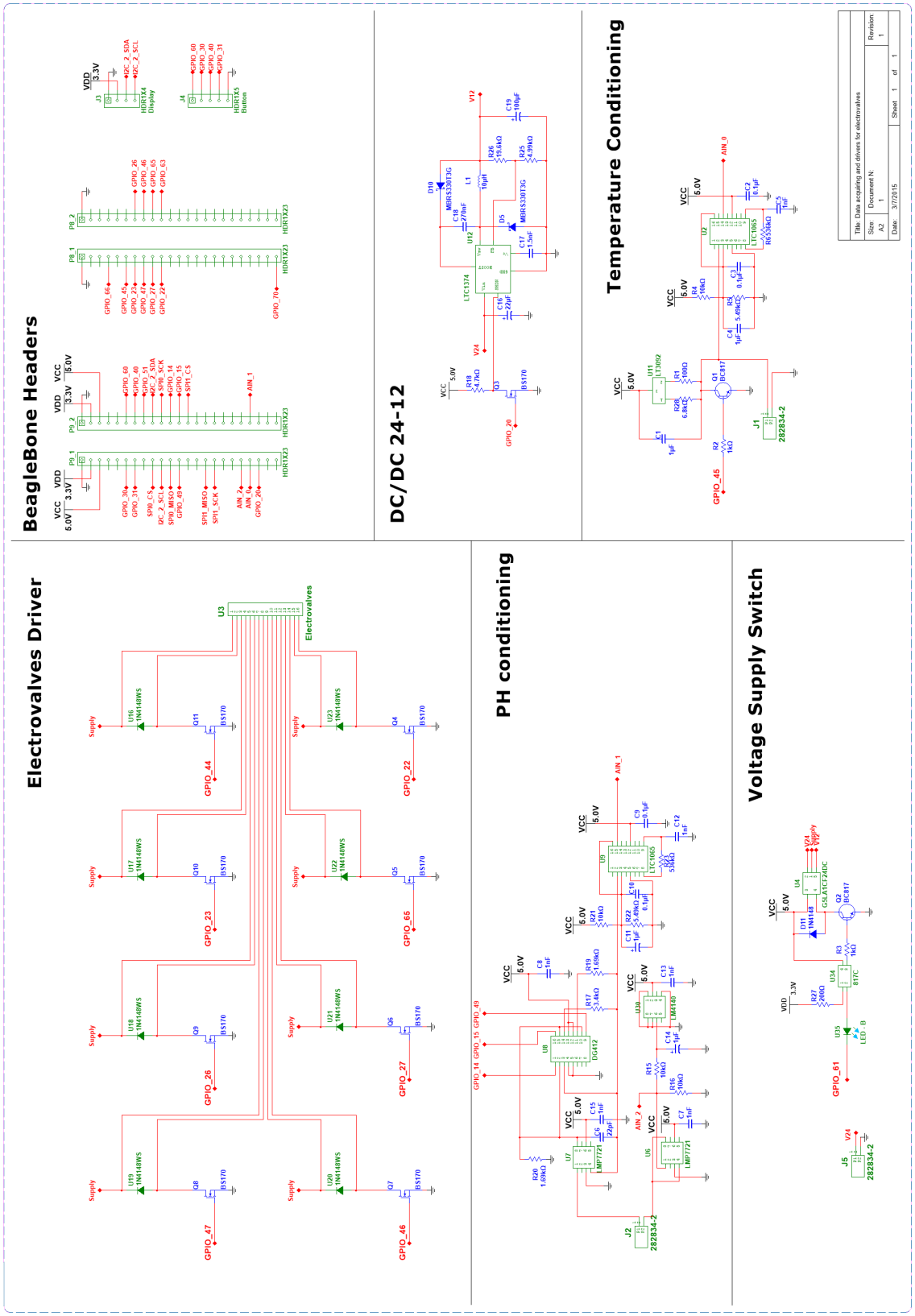


Figure 29: Schematic of Complete Circuit

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**.

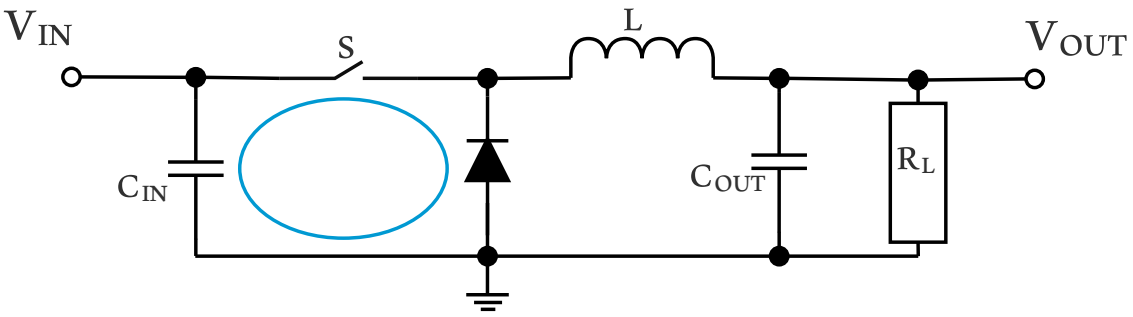


Figure 30: *DC-DC* High Speed Switching Path

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).

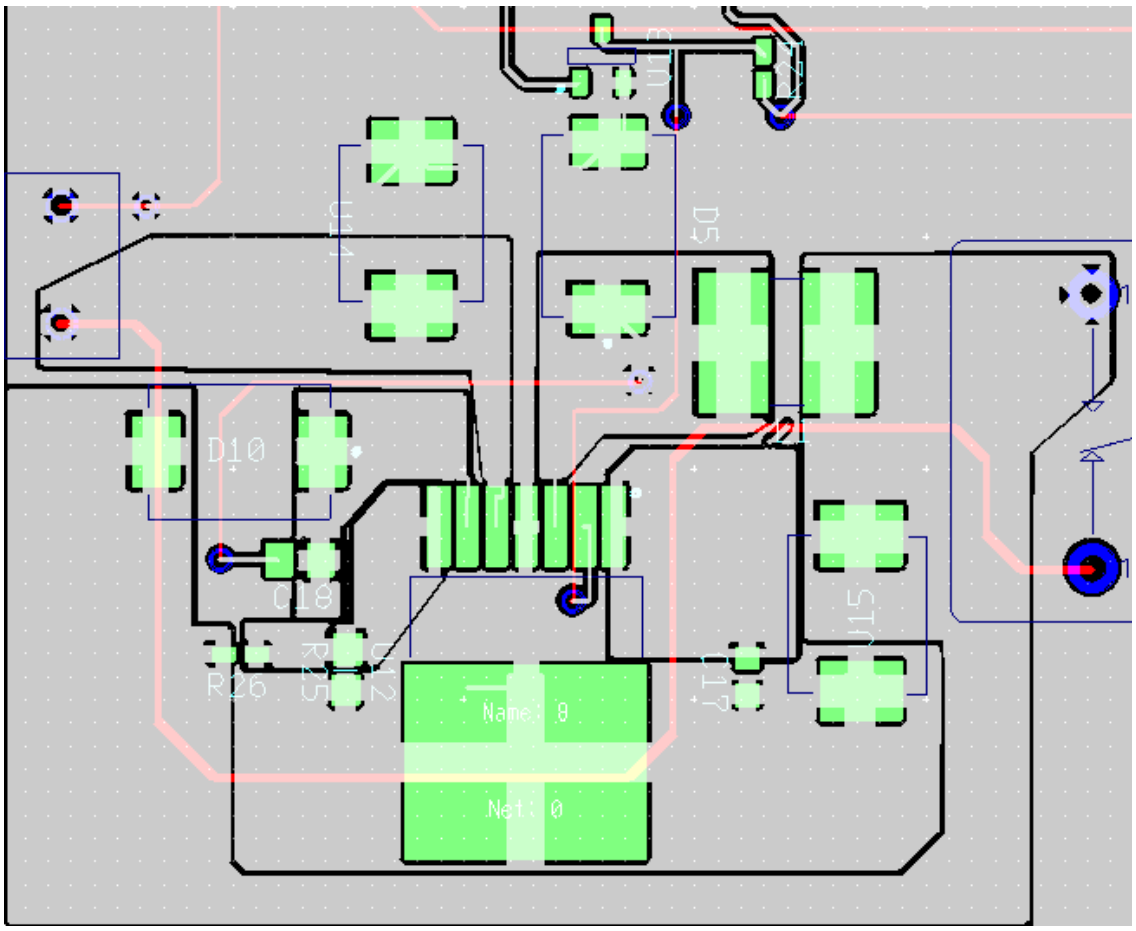


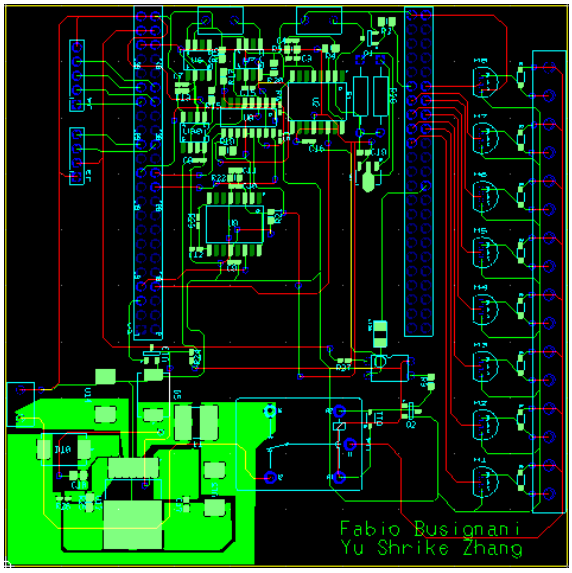
Figure 31: *PCB* Layout of *DC-DC*

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

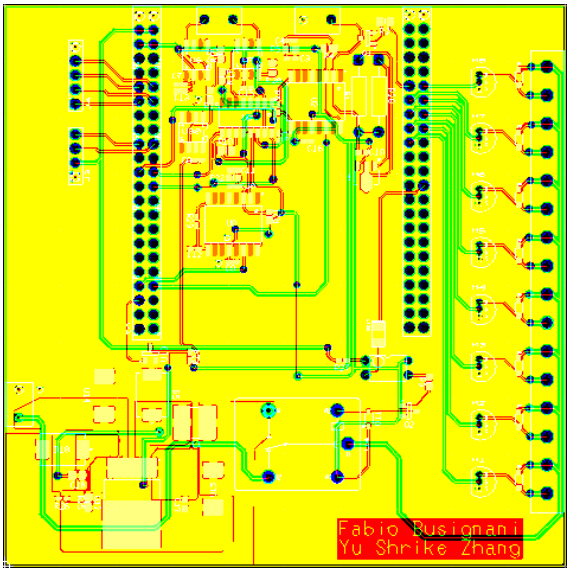
- the magnetic radiation is minimized by keeping by keeping catch diode and the input capacitor leads as short as possible;
- the electric radiation is minimized by reducing the area and length of all traces connected to the switch and boost pins.

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.

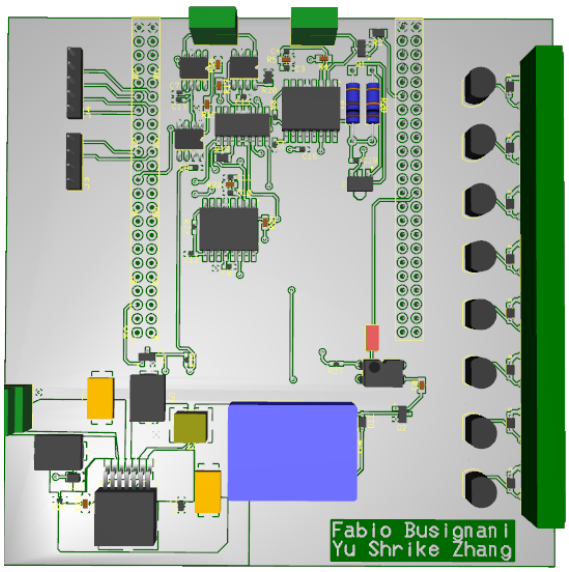
The (Fig.32) shows the *PCB* layout without ground plane (Fig.32a), then with both top and bottom ground plane (Fig.32b) which help with heat dissipation and *EMI* reduction. In (Fig.32c) and (Fig.32d) the 3-D model generated using *Ultiboard* and the final real result are compared.



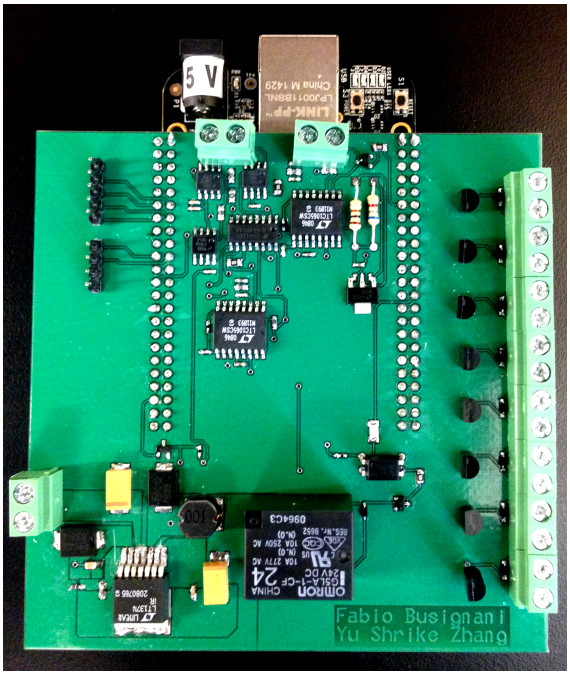
(a) *PCB* Layout Withot Ground Plane



(b) *PCB* Layout With Both Top and Bottom Ground Plane



(c) *PCB* 3-D Model View



(d) *PCB* Real Result

Figure 32: PCB of the System

Part II

FIRMWARE

3 INTRODUCTION

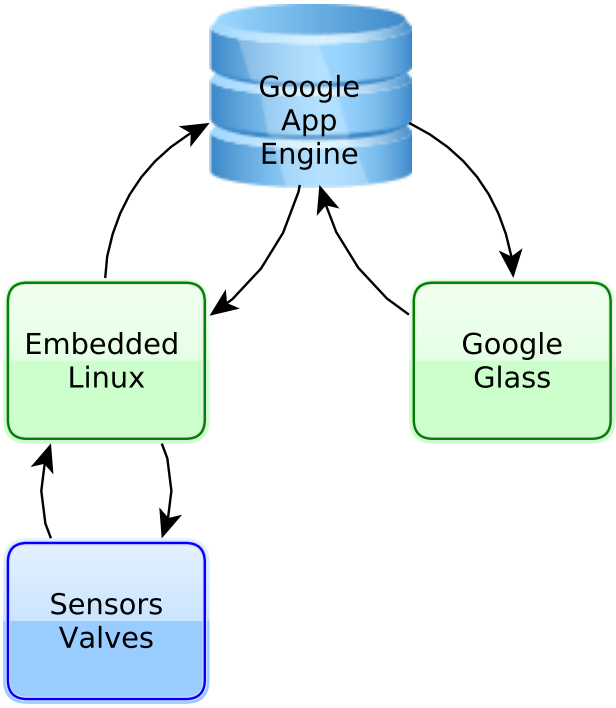


Figure 33: High Level System's Block Diagram

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 be 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..

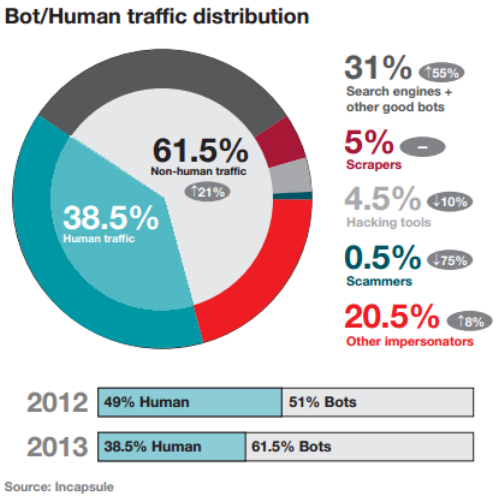


Figure 34: Distribution of Internet of Things

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

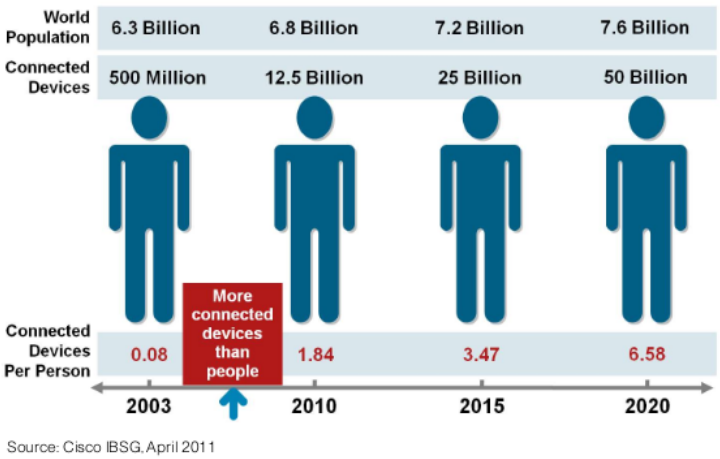


Figure 35: The *IoT* Evolution Prediction

3.2 THE BEAGLEBONE BLACK

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

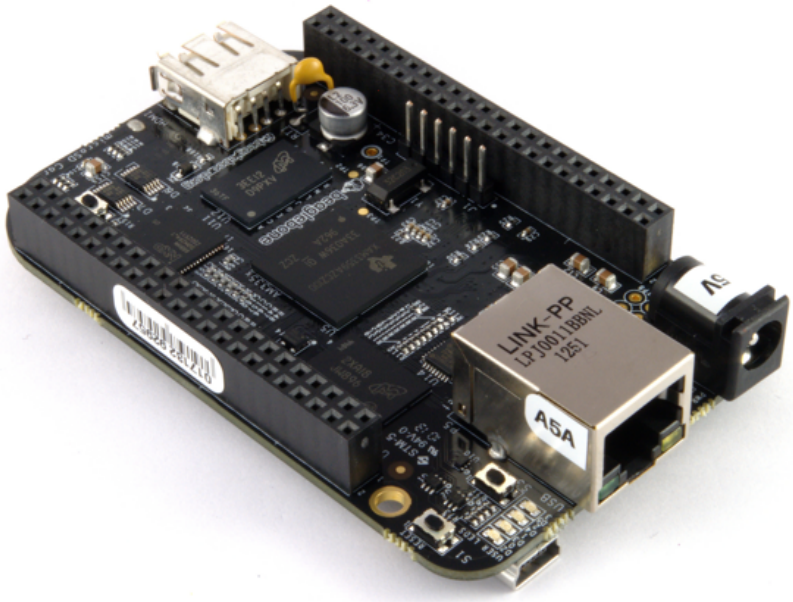


Figure 36: The *Beaglebone Black*

The *Beaglebone Black* is a low-cost, fully open-source (that means everyone can downloads and uses 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;
- $4GB$ 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;
- $512MB$ *DDR3* of system memory;
- Ethernet processor, which supports *DHCP*, so it can be directly conncted 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 o 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.

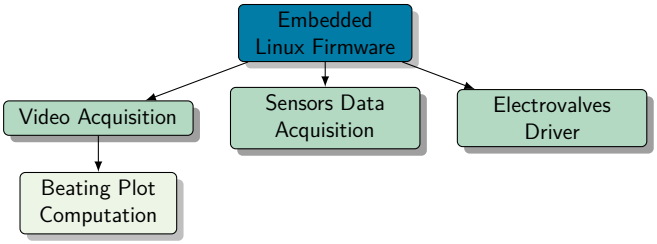


Figure 37: Embedded Linux Firmware - Blocks Subdivision

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™* microprocessor 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 is 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.

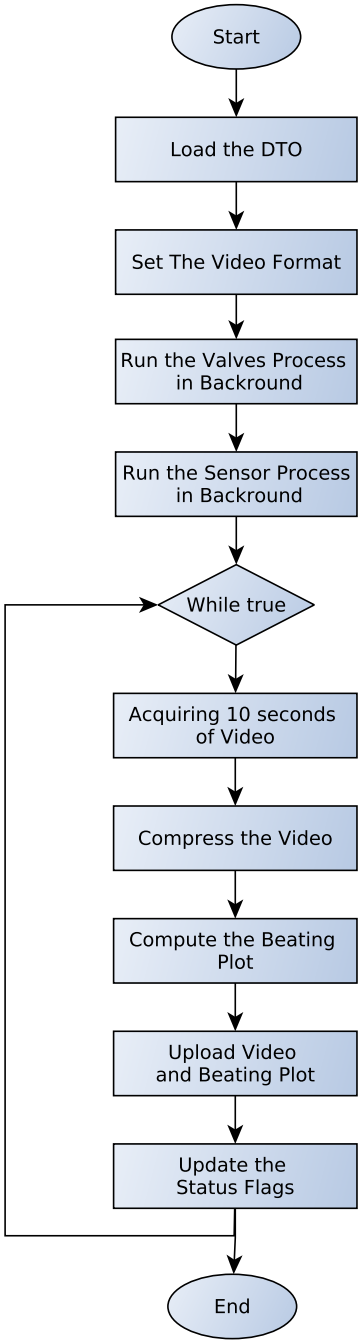


Figure 38: Bash Script Flow Chart

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 sores 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 has 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 value on the Datastore of Google App Engine then it waits for $100\ ms$. Two important features has to be highlighted:

1. as already introduced in (Sec.1.3), there is a *BJT* inside the temperature conditioning circuit that acts as software-controlled switch. This is necessary to avoid the self-heating of temperature sensor and ensure a correct measure. This software-controlling may be seen from the code, indeed there is a digital output called *temperature_disable* that is normally denied, and it is asserted only when a temperature measure has to be carried out;

2. as already introduced in (Sec.1.2), the pH measuring process need to know exactly how much is the amount of added offset. Indeed, in order to make the pH sensor response unipolar, the circuit polarizes the negative pin of pH sensor at approximately $512\ mV$. Instead of using the theoretical value it is better to measure it just before every pH measure process. This is easily viewable from the code.

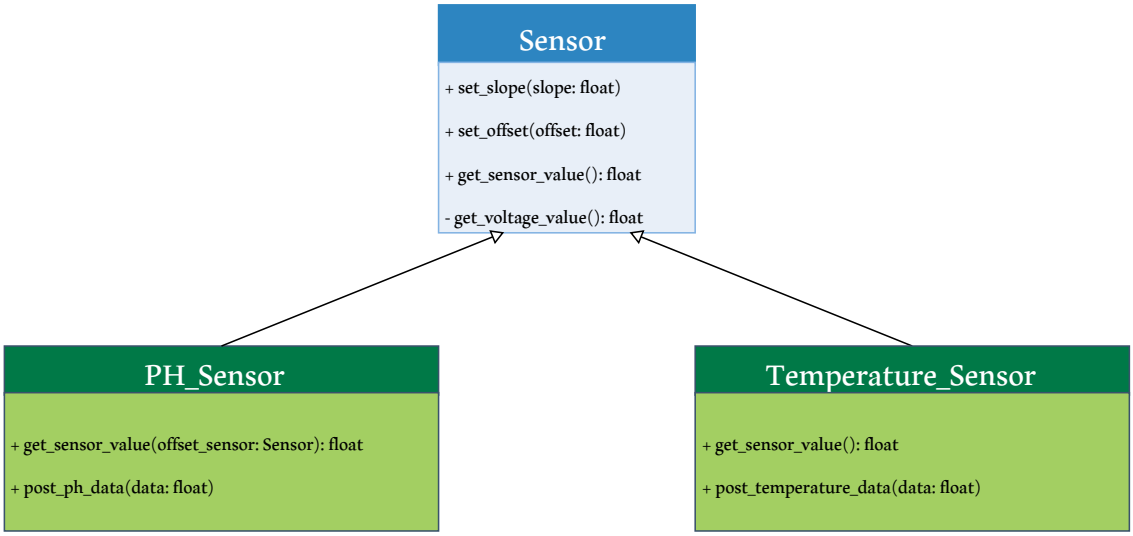


Figure 39: Sensor UML Description

4.3 ELECTROVALVES UPDATING TASK

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

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

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

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

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

Part III

SOFTWARE

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

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

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

5 GOOGLE APP ENGINE

Figure 40: Google App Engine Logo

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

Thus, when a programmer writes a web application that runs on App Engine, the software is going to run on the Google servers, somewhere in the Google cloude [6]. This solution is very useful because allows everyone to write the 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 website easy to manage. It has no database abstraction layer, indeed for this purpose, in this application, it supports the Google App Engine.

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

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

Figure 41: Flask Logo

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

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

5.1 THE WEB APPLICATION

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

1. *Sensor*, contains the list of sensors used from the system, through the method *sensor_list* it returns that list;

2. *DataPoint*, contains all the points that have to be plotted, the field value represent the $y - axes$ while time the $x - axes$. While the first field has to be passed at the moment in which the class is constructed, the *date* field is auto-filled with the current data. This class has two public methods to interact with:

   - *point_for_sensor*, that with the name of sensor as parameter returns the list of *DataPoint* associated;
   - *oldest_point*, that returns the last *DataPoint* inserted.

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

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

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

- *print_names*, available at the path `/sensor_names`. This function is accessible only through HTTP *GET* method and returns a *JSON* object with the content of sensor's name list.

- *graphing_data*, available at the path `/graphing_data`. To access to this function a HTTP *GET* method is required. This *GET* request must contain two parameters: *"sensor"*, the sensor name, and *"first_timestamp"*, which is the timestamp of the first point that has to be plotted. This last parameter is not mandatory and if it is missed it is assumed equal to 0. What this function does is to return a *JSON* object containing the whole DataPoint for the given sensor that have timestamp higher than *"first_timestamp"*.

- *clear_data*, available at the path `/clear`. This function is in charge to deletes all the points for each sensor.

- *set_picture*, available at path `/picture/submit`. This function is accessible with both HTTP *POST* and *GET* methods, and in both cases its behavior is to load a new image, containing the beating plot, onto Datastore. In the first case the image has be passed ad *POST* data, and named as *"Image.jpg"*. While th second case is supposed to be a browser way to upload the image. In fact, browsing to that *URL* what the user sees is shown in (Fig.42), a view where it is possible to select the picture file and submit it.

Figure 42: Picture Submitting from the Browser

- *view_picture*, available at the path `/picture/view`. Using this function, with a HTTP *GET* method causes the access to the image previously stored inside the Datastore.

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

- *view_video*, available at the path `/video/view`. This function, accessible with HTTP *GET* method, accesses to the microscope's video on the Datastore and returns it.

Figure 43: Video Submitting from the Browser

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

6 MICROSCOPE VIDEO STORING

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

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

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

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

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

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

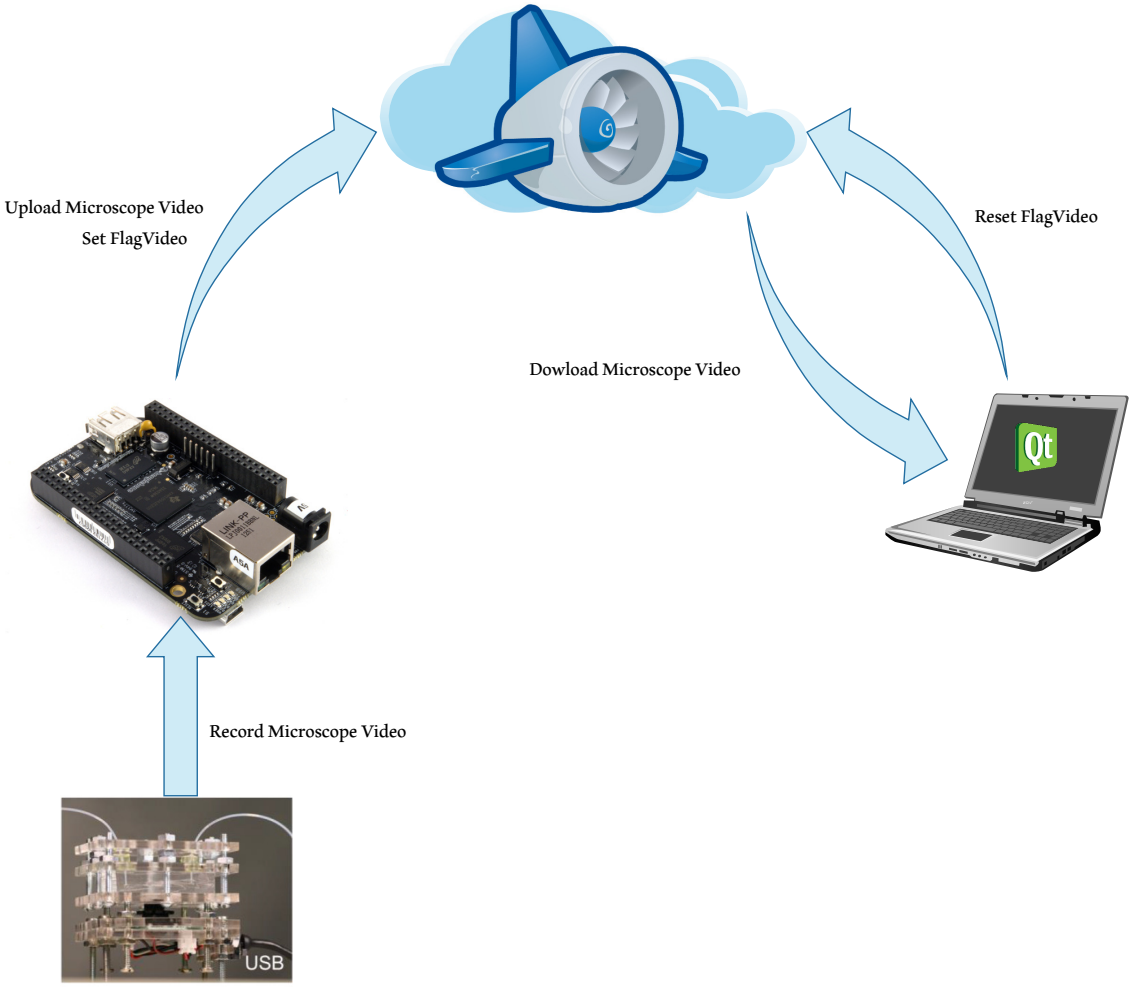


Figure 44: Video Storing: Block Diagram

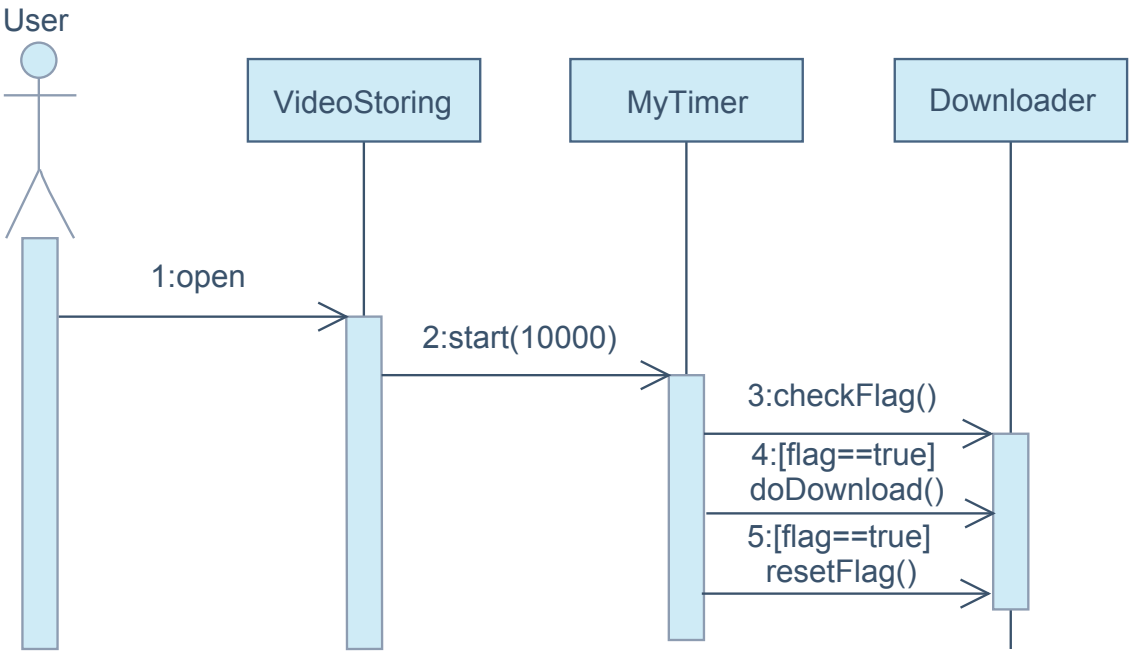


Figure 45: Video Storing: Sequence Diagram

Part IV

GOOGLE GLASS APPLICATION

7 WEARABLE COMPUTING - GOOGLE GLASS

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

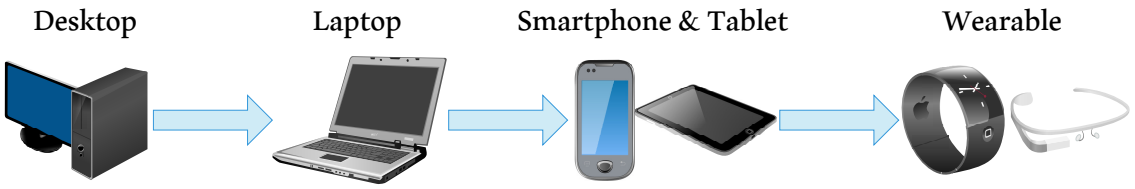


Figure 46: Computing Evolution

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

7.1 GOOGLE GLASS

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

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

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

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

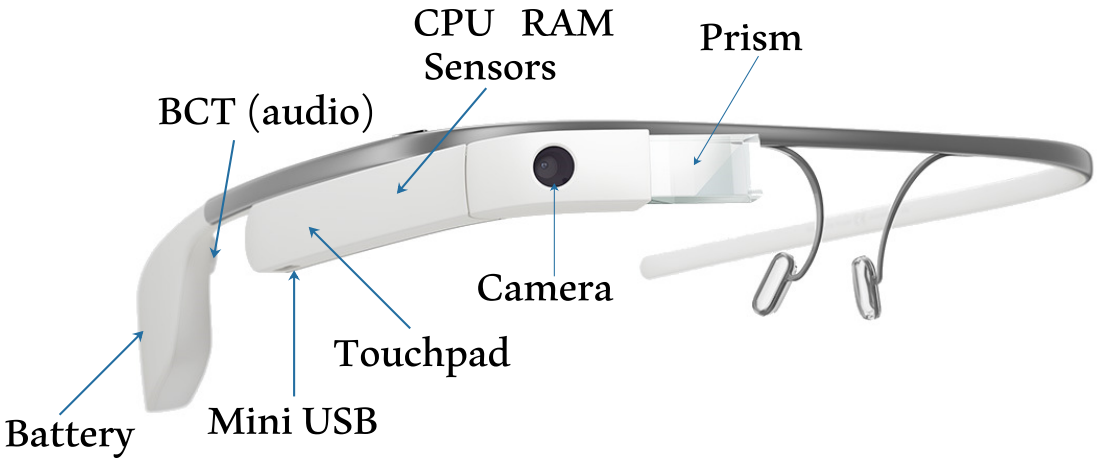


Figure 47: Google Glass

7.1.1 Mirror API

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

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

This service have two big requirements: of course, being connecting and cloud-aware. Mirror API Glassware has not executable file, and has no access to the Glass sensors.

7.1.2 Glass Development Kit

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

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

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

The Glassware designed in this thesis is GDK based.

8 THE GLASSWARE

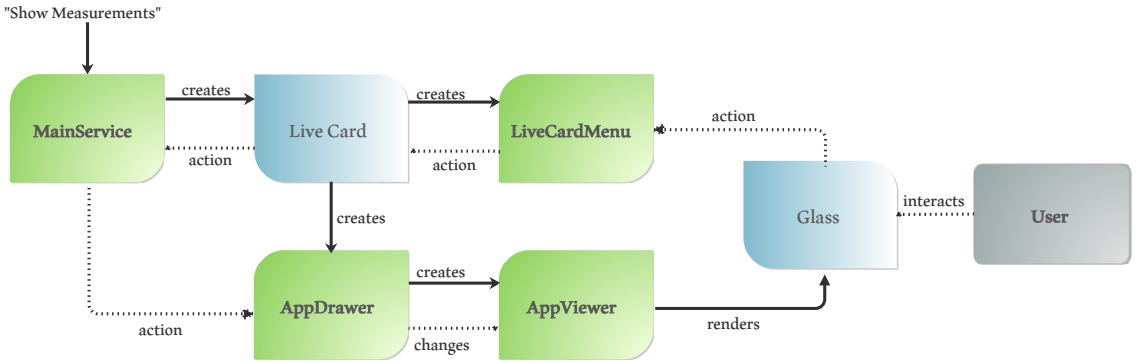


Figure 48: Interactions Between Class

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

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

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

Figure 49: MainService's Runnable in Background

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

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

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

Live cards are divided in two categories:

1. *Low-Frequency Rendering*, limited to a small set of views and can only update the display once every few seconds.

2. *High-Frequency Rendering*, involve content that is update frequently, more time in a second. This is very useful for plotting data coming from sensors, exactly like the application in which this Glassware is involved, this is why this kind of live card had been used in this thesis project. To create this type of live card an inflating layout has been used.

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

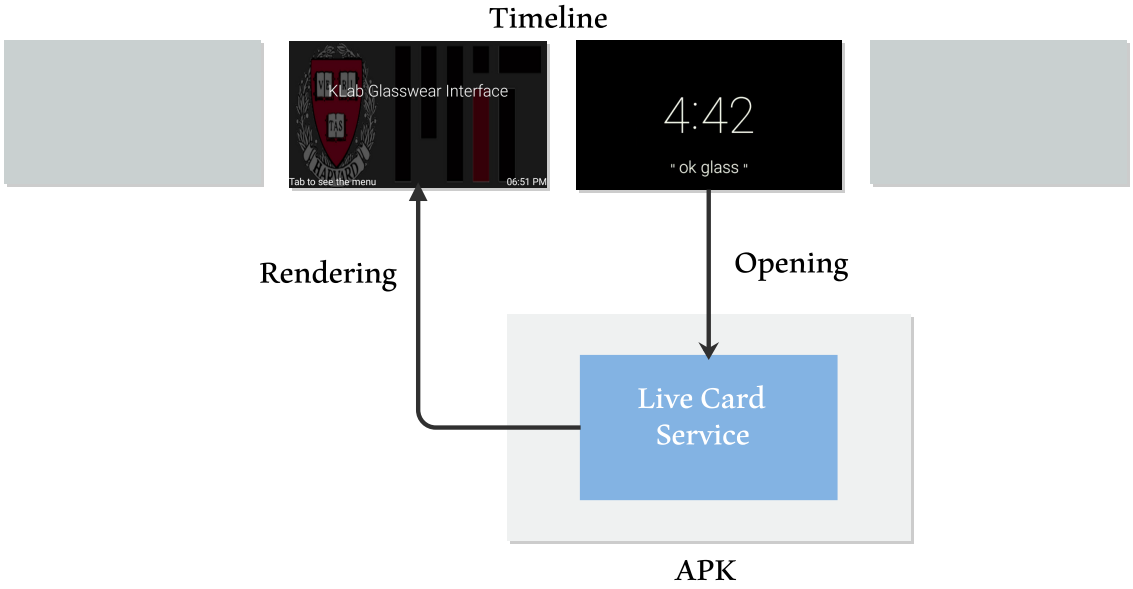


Figure 50: Live Card, Adding to the Timeline

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

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

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

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

8.1 THE CLASSES

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

8.1.1 Main Service

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

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

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

8.1.2 App Drawer

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

8.1.3 Menu Activity

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

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

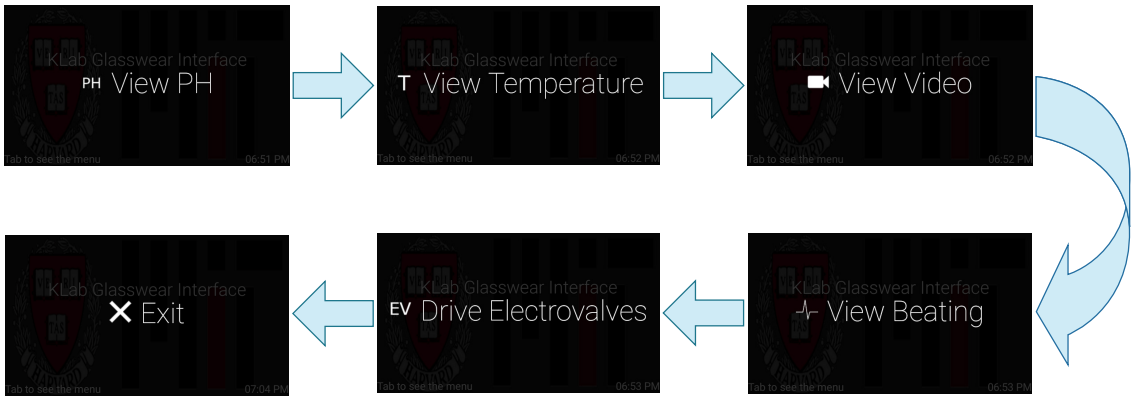


Figure 51: Main Menu

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

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

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

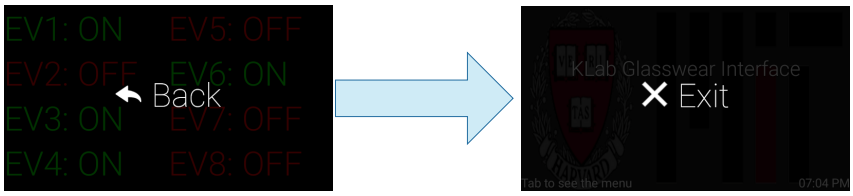


Figure 52: Graph Menu

Figure 53: Menu Electrovalves

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

8.1.4 App Manager

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

8.1.5 Main View

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

Figure 54: Main View

8.1.6 PH and Temperature Views

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

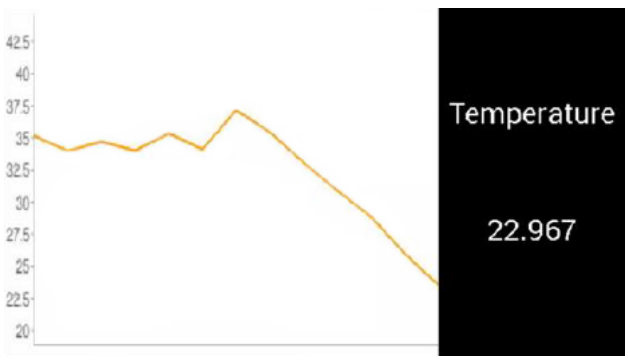


Figure 55: Temperature View

8.1.7 Beating View

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

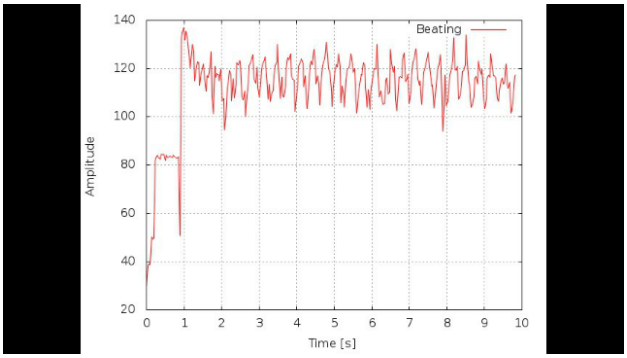


Figure 56: Beating View

8.1.8 Electrovalves View

(a) When Status Has Not Dowloaded Yet

(b) When They Are All Off

(c) When Some of Them is On

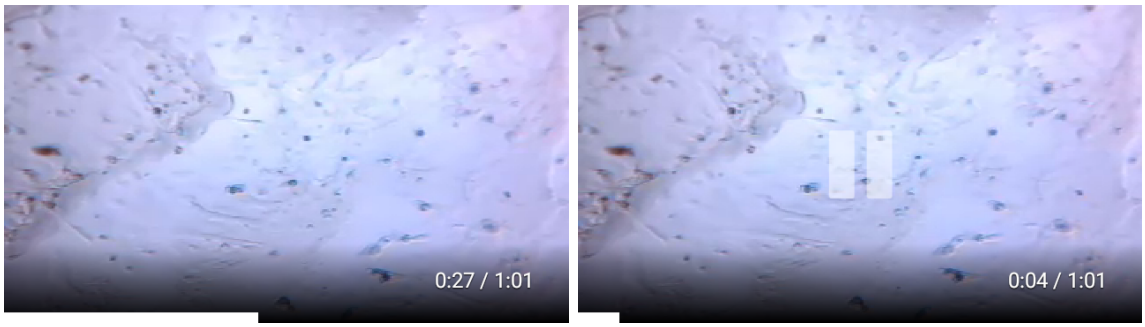
Figure 57: *Electrovalves View*

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

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

8.1.9 Video Activity

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



(a) Video **(b)** Video on Pause

Figure 58: *Microscope Video*

Part V

CONCLUSION AND APPENDIX

9 TEST AND PERFORMANCE

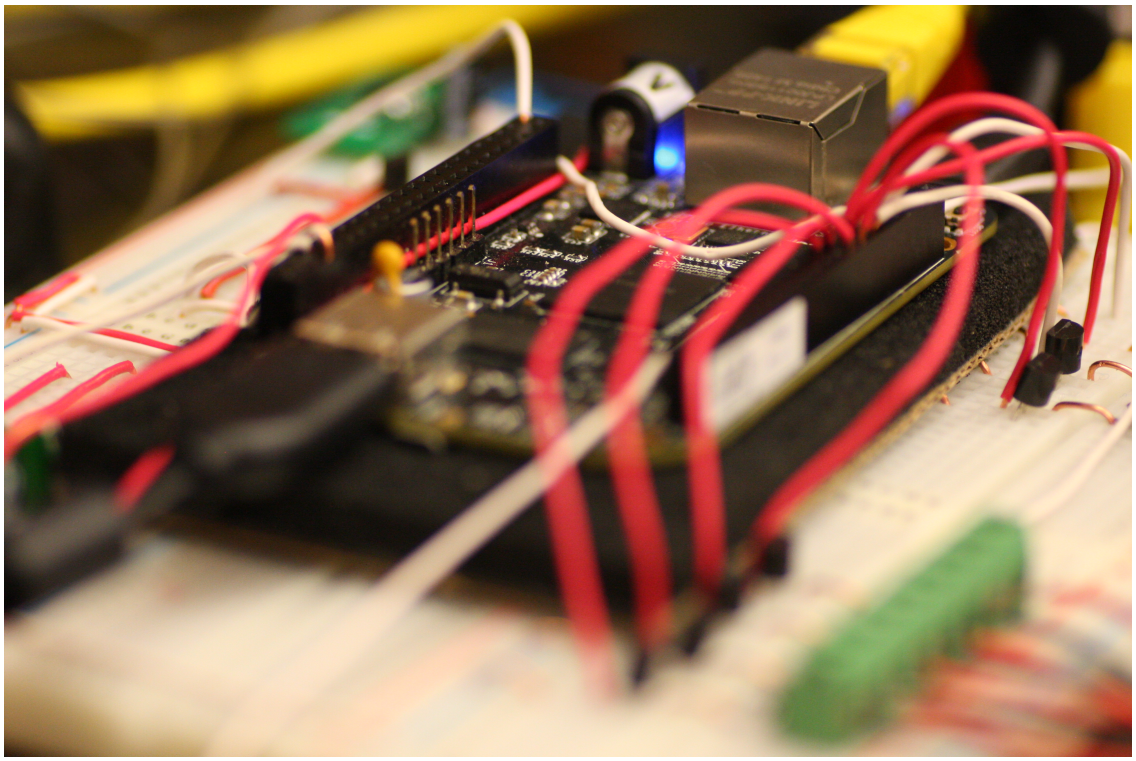


Figure 59: Circuit mounted on breadboard side view

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

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

9.1 LED EXPERIMENTS

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

going to work well with the electrovalves, too.

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

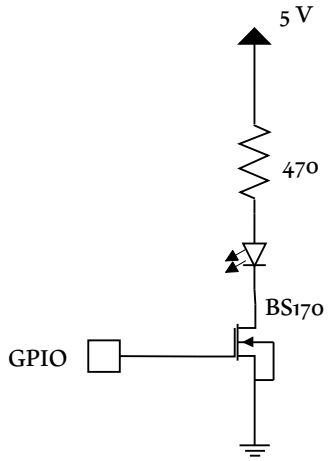


Figure 60: Driver for LED

The (Fig.61) shows the circuit used for this step of testing. As can be seen the number of LEDs used is eight, the same number of electrovalves that can be driven from this system.
In order to test all of them we made 2 different kind of trials:

1. *In order turning on&off*, first all the LEDs are turned on starting from the first one (on the top right corner) to the last one (on the bottom left corner). Then the LEDs are turned off following the same order.
   The result of this can bee watched in **this** video.

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

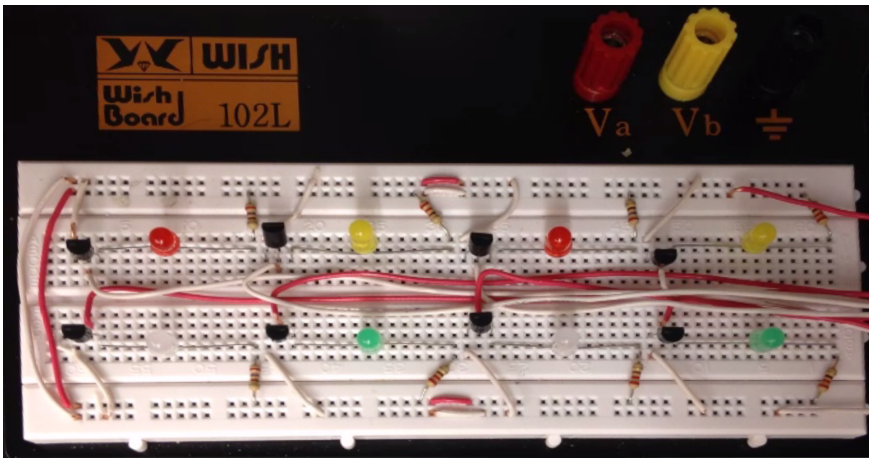


Figure 61: LEDs experiments board

9.2 ELECTROVALVES EXPERIMENTS

9.2.1 Breadboard Phase

The (Fig.62) shows from the top view the circuit used during the second phase of experiment, the one where we started using electrovalves in a real microfluidic application.
On the left side of the figure we can see the conditioning circuits for the temperature sensor (on the top) and pH sensor (on the bottom). While, on the other side, we can see the part of circuit in charge to drive the electrovalves.
In this last one we are going to focus for now. Each electrovalve is driven by the circuit shown in (Fig.23).

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

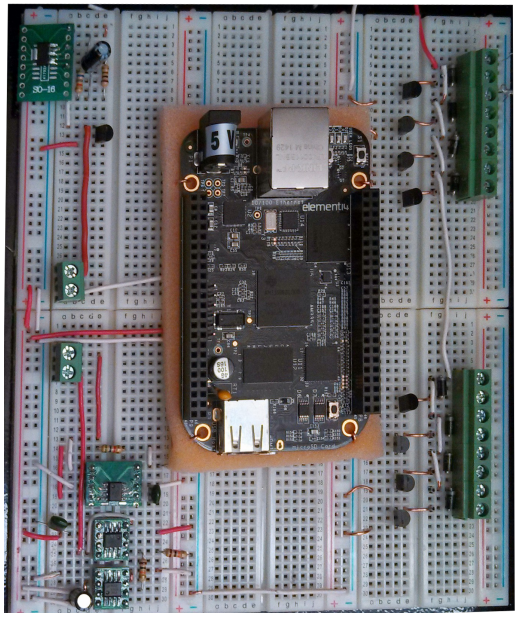


Figure 62: Circuit mounted on breadboard top view

The result of the experiments with electrovalves in a real microfluidic case can bee watched in **this** video.

9.2.2 PCB Phase

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

10 LIMITATIONS AND IMPROVEMENTS

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

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

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

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

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

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

A CODE

A.1 FIRMWARE

Listing A.1: KlabFirware (bash)

```bash
#!/bin/bash
echo "Load the DTO"
echo GlassProject-GPIO > $SLOTS

echo "Setting the video format as MJPG"
v4l2-ctl --set-fmt-video=width=320,height=240,pixelformat=1 -d 0
v4l2-ctl --set-parm=30

echo "Running the Electrovalve status updating task"
./Electrovalves/ValvesUpdating </dev/null &>/dev/null &
./Sensors/sensorAcquiring </dev/null &>/dev/null &

while true; do
  echo "Recording ten second of Video"
  ./Video/capture -d /dev/video0 -F -o -c 300 > output.raw -y
  avconv -f mjpeg -i output.raw -vcodec copy output.mp4 -y
  echo "Compressing Video"
  ffmpeg -i output.mp4 -acodec mp2 Video.mp4
  echo "Uploading the Video to the server"
  curl -include --form Video.mp4 =@Video.mp4 -A "National Instruments LabVIEW"
      http://CENSURED/video/submit -0
  echo "Updating the flag for video"
  curl  --data "name=video&status=on"
      http://CENSURED/add/electrovalve
  curl  --data "name=glass&status=on"
      http://CENSURED/add/electrovalve
  echo "Ploting the beating rate"
  ./Video/compute_beating
  echo "Uploading the beating plot"
  curl -include --form Image.jpg=@Image.jpg -A "National Instruments LabVIEW"
      http://CENSURED/picture/submit -0
  echo "finish"
done
```

Listing A.2: SensorAcquiring (bash)

```
#!/bin/bash

while true; do
  echo "Clearing previus data on server"
  curl http://CENSURED/clear
  echo "Acquiring and uploading new sensors data"
  ./sensor_acquiring

done
```

Listing A.3: Pins Setting

```
/*
* Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Purpose License Version 2 as
* published by the Free Software Foundation
*
* Original from: github.com/jadonk/validation-scripts/blob/master/test-capemgr/
*
* Modified by Fabio Busignani fbusigna@mit.edu
*
*/

/dts-v1/;
/plugin/;

/{
        compatible = "ti,beaglebone", "ti,beaglebone-black";
        part-number = "KLab";
        version = "00A0";

        fragment@0 {
              target = <&am33xx_pinmux>;

              __overlay__ {
                   ebb_example: KLab {
                          pinctrl-single,pins = <
                                0x04c 0x07  // P9_16 - T - Output Mode7 pulldown
                                0x024 0x07  // P8_13 - EV1 - Output Mode7 pulldown
                                0x028 0x07  // P8_14 - EV4 - Output Mode7 pulldown
                                0x03c 0x07  // P8_15 - EV3 - Output Mode7 pulldown
                                0x038 0x07  // P8_16 - EV6 - Output Mode7 pulldown
                                0x02c 0x07  // P8_17 - EV5 - Output Mode7 pulldown
                                0x08c 0x07  // P8_18 - EV8 - Output Mode7 pulldown
                                0x020 0x07  // P8_19 - EV7 - Output Mode7 pulldown
                                0x030 0x07  // P8_44 - EV2 -Output Mode7 pulldown
```

```
                                0x0a0 0x07   // P8_46 - PowerSupply


                            >;
                    };
                };
        };

        fragment@1 {
    target = <&ocp>;
    __overlay__ {
      gpio_helper {
        compatible = "gpio-of-helper";
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&eklab>;
      };
    };
  };
};
```

Listing A.4: ElectrovalvesDriver.cpp

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <curl/curl.h>
#include <GPIO/GPIO.h>

#define NUMBER_ELECTROVALVES 8

using namespace std;

std::string const ELECTROVALVES_LINK =
    "http://CENSURED/show/";
std::string const TRUE_VALUE = " True ";
std::string const FALSE_VALUE = " False ";

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

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

```
  GPIO DCDCENABLE(20);

  CURL *curl;
  CURLcode res;

  GPIO_VALUE output_vector[NUMBER_ELECTROVALVES];

  //Set output pins
  EV1.setDirection(OUTPUT);
  EV2.setDirection(OUTPUT);
  EV3.setDirection(OUTPUT);
  EV4.setDirection(OUTPUT);
  EV5.setDirection(OUTPUT);
  EV6.setDirection(OUTPUT);
  EV7.setDirection(OUTPUT);
  EV8.setDirection(OUTPUT);

  RELAY.setDirection(OUTPUT);

  if (argc == 5)
  {
    if(atoi(argv[1])==24)
    {
        RELAY.setValue(HIGH);
        DCDCENABLE.setValue(HIGH);//disable the DCDC
    }
    else
    {
        RELAY.setValue(LOW);
        DCDCENABLE.setValue(LOW);//enable the DCDC
    }
  }
  else
  {
    RELAY.setValue(HIGH);
    DCDCENABLE.setValue(HIGH);//disable the DCDC
  }


  while(1)
  {
    //acquire the values of electrovalves status and store them inside
    //output_vector
    int i;
    for(i=0;i<NUMBER_ELECTROVALVES; i++)
    {
      curl = curl_easy_init();
      if(curl)
      {
```

```
        std::string readBuffer;
        std::ostringstream ss;
        ss << i+1;
        std::string url = ELECTROVALVES_LINK + "EV" + ss.str();
        curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
        /* example.com is redirected, so we tell libcurl to follow redirection */
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
        /* Perform the request, res will get the return code */
        res = curl_easy_perform(curl);
        /* always cleanup */
        curl_easy_cleanup(curl);

        if(readBuffer.compare(TRUE_VALUE) == 0)
        {
          std::cout << "EV" + ss.str()+": ON" << std::endl;
          output_vector[i] = HIGH;
        }
        else if(readBuffer.compare(FALSE_VALUE) == 0)
        {
          std::cout << "EV" + ss.str()+": OFF" << std::endl;
          output_vector[i] = LOW;
        }
      }
    }
    //now I'm ready to write the output pins:
    EV1.setValue(output_vector[0]);
    EV2.setValue(output_vector[1]);
    EV3.setValue(output_vector[2]);
    EV4.setValue(output_vector[3]);
    EV5.setValue(output_vector[4]);
    EV6.setValue(output_vector[5]);
    EV7.setValue(output_vector[6]);
    EV8.setValue(output_vector[7]);
  }
  return 0;
}
```

Listing A.5: sensor_aquiring.cpp

```
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<string>
#include<sstream>
#include<stdlib.h>
#include<curl/curl.h>
#include"../GPIO.h"
#include"../http.h"
```

```
using namespace std;
#define LDR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"
#define GPIO_PATH "/sys/class/gpio/"
#define OUTPUT false
#define INPUT true
#define HIGH true
#define LOW false

float const CURRENT = 0.00068;
float const SLOPE_TEMPERATURE = 2;
float const OFFSET_TEMPERATURE = 1870;
float const SLOPE_PH = -0.070;
float const OFFSET_PH = 0.6;

class Sensor
{
  int number;


public:
  float m;
  float q;
  Sensor(int x): number(x){}
  void set_slope(float slope){
    m = slope;
  }
  void set_offset(float offset){
    q = offset;
  }
  float get_sensor_value(){
    float voltage_value = this->get_voltage_value();
    float sensor_value = (voltage_value - q)/m;
    return sensor_value;
  }
private:
  float get_voltage_value(){
    stringstream ss;
    ss << LDR_PATH << number << "_raw";
    fstream fs;
    int adc_value;
    fs.open(ss.str().c_str(), fstream::in);
    fs >> adc_value;
    fs.close();
    float cur_voltage = adc_value * (1.80f/4096.0f);
    float actual_value;
    return cur_voltage;

  }
};
```

```
59
   class Temperature_Sensor : public Sensor{
61 private:

63 public:
     Temperature_Sensor(int x) : Sensor(x) {}
65   float get_sensor_value(){
       float voltage_value = this->get_voltage_value();
67     float resistor_value = voltage_value / CURRENT;
       float sensor_value = (resistor_value - q)/m;
69     return sensor_value;
     }
71   void post_temperature_data(float data){
       post_data_sensor(0, data);
73   }
   };
75
   class PH_Sensor : public Sensor{
77
   public:
79   PH_Sensor(int x) : Sensor(x) {}
     float get_sensor_value(Sensor offset_sensor){
81       //Sensor offset_sensor(2);
         float voltage_value = this->get_voltage_value();
83       q += offset_sensor.get_voltage_value();
         float sensor_value = (voltage_value - q)/m;
85       return sensor_value;
       }
87
     void post_ph_data(float data){
89     post_data_sensor(1, data);
     }
91 };

93 int main(int argc, char* argv[])
   {
95   GPIO temperature_disable(45);
     float slope_temperature;
97   float slope_ph;
     float offset_temperature;
99   float offset_ph;

101  if (argc == 5)
     {
103    slope_temperature = atoi(argv[1]);
       offset_temperature = atoi(argv[2]);
105    slope_ph = atoi(argv[3]);
       offset_ph = atoi(argv[4]);
107  }
```

```
  else
  {
    slope_temperature = SLOPE_TEMPERATURE;
    offset_temperature = OFFSET_TEMPERATURE;
    slope_ph = SLOPE_PH;
    offset_ph = OFFSET_PH;
  }

  //set the pin GPIO_45 as output
  temperature_disable.setDirection(OUTPUT);
  //Temporary disable the temperature
  temperature_disable.setValue(HIGH);

  Temperature_Sensor temperature_sensor(2);
  PH_Sensor ph_sensor(1);
  Sensor offset_sensor(2);

  temperature_sensor.set_slope(slope_temperature);
  temperature_sensor.set_offset(offset_temperature);
  ph_sensor.set_slope(slope_ph);
  ph_sensor.set_offset(offset_ph);

  int i= 0;
  for(i=0; i<100; i++)
  {
    temperature_disable.setValue(LOW);
    float ph = ph_sensor.get_sensor_value(offset_sensor);
    float temperature = temperature_sensor.get_sensor_value();
    temperature_disable.setValue(HIGH);
    cout << "The voltage value is: " << temperature << " V." << endl;
    temperature_sensor.post_temperature_data(temperature);
    ph_sensor.post_ph_data(ph);
    usleep(100000);
  }
  float temperature = temperature_sensor.get_voltage_value();
  cout << "The voltage value is: " << temperature << " V." << endl;
  return 0;
}
```

Listing A.6: compute_beating.cpp

```

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

```
#include<opencv2/opencv.hpp>   // C++ OpenCV include file
using namespace std;
using namespace cv;            // using the cv namespace too

int main()
{
    ofstream point_file;
    point_file.open("video_data.dat");
    VideoCapture capture; // capturing from file
    capture.open("output.mp4");

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

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

    Mat frame, firstFrame, diffFrame;

    capture >> frame; //save the first frame
    firstFrame = frame;
    Scalar color_information = mean(firstFrame);
    float initial_point = (color_information[0] +color_information[1]
      +color_information[2])/3;

    int i;
    for(i=1; i<capture.get(CV_CAP_PROP_FRAME_COUNT);i++)
    {
        capture >> frame;          // capture the image to the frame
        if(frame.empty())
        {
            cout << "Failed to capture an image" << endl;
            return -1;
        }
        absdiff(frame, firstFrame, diffFrame);
        color_information = mean(frame);
        float red = color_information[0];
        float green = color_information[1];
        float blue = color_information[2];

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

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

```

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

61      }

63

    point_file.close();
65    return 0;
}
```

Listing A.7: Pins Setting

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

14 /dts-v1/;
 /plugin/;
16
 /{
18        compatible = "ti,beaglebone", "ti,beaglebone-black";
          part-number = "KLab";
20        version = "00A0";

22        fragment@0 {
                target = <&am33xx_pinmux>;
24
                __overlay__ {
26                   ebb_example: KLab {
                             pinctrl-single,pins = <
28                                 0x04c 0x07  // P9_16 - T - Output Mode7 pulldown
                                   0x024 0x07  // P8_13 - EV1 - Output Mode7 pulldown
30                                 0x028 0x07  // P8_14 - EV4 - Output Mode7 pulldown
                                   0x03c 0x07  // P8_15 - EV3 - Output Mode7 pulldown
32                                 0x038 0x07  // P8_16 - EV6 - Output Mode7 pulldown
                                   0x02c 0x07  // P8_17 - EV5 - Output Mode7 pulldown
34                                 0x08c 0x07  // P8_18 - EV8 - Output Mode7 pulldown
                                   0x020 0x07  // P8_19 - EV7 - Output Mode7 pulldown
36                                 0x030 0x07  // P8_44 - EV2 -Output Mode7 pulldown
                                   0x0a0 0x07  // P8_46 - PowerSupply
```

```
38
                            >;
40                      };
                    };
42          };

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

Listing A.8: HTTP.h

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

11 std::string const URL_POST = "http://CENSURED/sensor_values";
   std::string const POST_DATA_TEMPERATURE =  "sensor=temperature&value=";
13 std::string const POST_DATA_PH =  "sensor=pH&value=";
   std::string const POST_DATA_BEATING =  "sensor=beating&value=";
15


17 void post_data_sensor(int type, float value);

19 #endif // HTTP
```

Listing A.9: HTTP.cpp

```
  #include "http.h"
2

  void post_data_sensor(int type, float value){
4         CURL *curl;
          CURLcode res;
6         /* 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());
     /* Perform the request, res will get the return code */
          res = curl_easy_perform(curl);
          /* Check for errors */
          if(res != CURLE_OK)
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
                    curl_easy_strerror(res));
    /* always cleanup */
          curl_easy_cleanup(curl);
        }
        curl_global_cleanup();
}
```

A.2 VIDEO STORING SOFTWARE

Listing A.10: main.cpp

```
#include <QCoreApplication>
#include "downloader.h"
#include "mytimer.h"
#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    MyTimer t;
    return a.exec();
```

```
}
```

Listing A.11: VideoStoring.pro

```
#-------------------------------------------------
#
# Project created by QtCreator 2015-02-17T21:56:37
#
#-------------------------------------------------

QT       += core
QT       += network

QT       -= gui

#RC_FILE = myapp.rc

TARGET = VideoStoring
CONFIG   += console
CONFIG   -= app_bundle

TEMPLATE = app


SOURCES += main.cpp \
    downloader.cpp \
    mytimer.cpp

HEADERS += \
    downloader.h \
    mytimer.h
```

Listing A.12: mytimer.h

```
#ifndef MYTIMER_H
#define MYTIMER_H

#include <QObject>
#include <QtCore>
#include <downloader.h>

class MyTimer : public QObject
{
    Q_OBJECT
public:
    MyTimer();
    ~MyTimer();
    QTimer *timer;

```

```
17 public slots:
       void timerSlot();
19 private:
       Downloader d;
21 };

23 #endif // MYTIMER_H
```

Listing A.13: mytimer.cpp

```
1  #include "mytimer.h"
   #include <QtCore>
3  #include <QDebug>
   MyTimer::MyTimer()
5  {
       timer = new QTimer(this);
7      connect(timer,SIGNAL(timeout()), this, SLOT(timerSlot()));

9      timer->start(10000);

11 }

13 MyTimer::~MyTimer()
   {
15 }

17 void MyTimer::timerSlot(){
       d.checkFlag();
19     if( d.getStatus())
       {
21         d.resetFlag();
           d.doDownload();
23     }

25 }
```

Listing A.14: downloader.h

```
   #ifndef DOWNLOADER_H
2  #define DOWNLOADER_H

4  #include <QObject>
   #include <QNetworkAccessManager>
6  #include <QNetworkRequest>
   #include <QNetworkReply>
8  #include <QUrl>
   #include <QDateTime>
10 #include <QFile>
```

```
#include <QDebug>
#include <QEventLoop>

class Downloader : public QObject
{
    Q_OBJECT
public:
    explicit Downloader(QObject *parent = 0);

    void doDownload();
    void checkFlag();
    bool getStatus();
    void resetFlag();

signals:

public slots:
    void replyFinished (QNetworkReply *reply);
    void checkStatus(QNetworkReply *replay);

private:
   QNetworkAccessManager *manager;
   QNetworkAccessManager *manager2;
   QDateTime data;
   QString name;
   bool flag_ready;
   bool status;


};

#endif // DOWNLOADER_H
```

Listing A.15: downloader.cpp

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

Downloader::Downloader(QObject *parent) :
    QObject(parent)
{
}

void Downloader::doDownload()
{
    manager = new QNetworkAccessManager(this);

    connect(manager, SIGNAL(finished(QNetworkReply*)),
            this, SLOT(replyFinished(QNetworkReply*)));
```

```
16      manager->get(QNetworkRequest(
                QUrl("http://CENSURED/video/view")));
18  }

20  void Downloader::replyFinished (QNetworkReply *reply)
    {
22      if(reply->error())
        {
24          qDebug() << "ERROR!";
            qDebug() << reply->errorString();
26      }
        else
28      {
            qDebug() << reply->header(QNetworkRequest::ContentTypeHeader).toString();
30          qDebug() << reply->header(QNetworkRequest::LastModifiedHeader).
                  toDateTime().toString();
32          qDebug() << reply->header(QNetworkRequest::ContentLengthHeader).
                  toULongLong();
34          qDebug() << reply->attribute(QNetworkRequest::HttpStatusCodeAttribute).
                  toInt();
36          qDebug() << reply->attribute(QNetworkRequest::HttpReasonPhraseAttribute).
                  toString();
38
            data = QDateTime::currentDateTime();
40          name = "C:/Video/"+data.toString("yyyy-MM-dd-HH-mmss")+".mpeg";

42          QFile *file = new QFile(name);
            if(file->open(QFile::Append))
44          {
                file->write(reply->readAll());
46              file->flush();
                file->close();
48          }
            delete file;
50      }

52      reply->deleteLater();
    }
54

56  void Downloader::checkStatus(QNetworkReply *replay)
    {
58      QString result(replay->readAll());
        int compare = QString::compare(result, FLAG_ON);
60      if(compare == 0)
        {
62          status = true;
        }
```

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

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

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

A.3 GOOGLE APP ENGINE

Listing A.16: Main Script of Server

```
from flask import Flask

app = Flask(__name__)

from flask import request
from flask import make_response

from models import MESSAGES
from google.appengine.ext import ndb
from google.appengine.api import memcache
import logging
import datetime
import cloudstorage
import json

SECONDS_BETWEEN_UPDATES = 60
SENSOR_TIMEOUT_IN_SECONDS = 300
ELECTROVALVES_TIME_OUT = 1000
BUCKET_NAME = "/CENSURED/"
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'mp4'}
UPDATE_FLAG = 'database_updated_recently'

#DataPoint contains all the points that have to be plotted
class DataPoint(ndb.Model):
    #the y-axes value
    value = ndb.FloatProperty()
    #the x-axes value (autofilled with the current data)
    date = ndb.DateTimeProperty(auto_now_add=True)

    @classmethod
    # returns the list of DataPoint associated with sensor_key
    def points_for_sensor(cls, sensor_key):
        return cls.query(ancestor=sensor_key).order(-cls.date)

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

#Sensor class contains the list of used sensors
class Sensor(ndb.Model):
    @classmethod
    # returns that list
    def sensor_list(cls):
```

```
45          return cls.query()


47
def update():
49      # Check to see if we have updated recently
        update_flag = memcache.get(UPDATE_FLAG)
51      if update_flag is not None:
            return
53      memcache.add(UPDATE_FLAG, 'YES', SECONDS_BETWEEN_UPDATES)
        # Delete old data points
55      now = datetime.datetime.now()
        count = 0
57      for point in DataPoint.oldest_points().iter():
            delta = (now - point.date).total_seconds()
59          if delta > 3600:
                point.key.delete()
61              count += 1
            else:
63              break
        logging.info("Deleted {} old data points".format(count))
65      # Loop through all of the sensors
        sensor_names = []
67      for sensor in Sensor.sensor_list().iter():
            # Store current values of each sensor
69          sensor_value = memcache.get(sensor.key.id() + "_value")
            if sensor_value is not None:
71              point = DataPoint(parent=sensor.key, value=sensor_value)
                point.put()
73          # Delete sensors that don't have any data
            curr_points = DataPoint.points_for_sensor(sensor.key)
75          if curr_points.count() == 0 and sensor_value is None:
                logging.info("Deleting sensor {}".format(sensor.key.id()))
77              sensor.key.delete()
            else:
79              sensor_names.append(sensor.key.id())
        # Store a list of sensor names in memcache
81      memcache.set("sensor_names", ",".join(sensor_names))

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

```
        if sensor_names is None:
            sensor_names = ",".join([sensor.key.id() for sensor in
             Sensor.sensor_list()])
            memcache.set('sensor_names', sensor_names)
        sensor_names = sensor_names.split(",")
        sensor_values = {}
        for sensor_name in sensor_names:
            sensor_value = memcache.get(sensor_name + '_value')
            if sensor_value is not None:
                sensor_values[sensor_name] = sensor_value
        return json.dumps(sensor_values)
    elif request.method == 'POST':
        update()
        logging.info(request.values)
        sensor_name = request.form.get('sensor')
        sensor_value = float(request.form.get('value'))
        sensor_names = memcache.get('sensor_names')
        if sensor_names is not None and sensor_name not in sensor_names:
            sensor = Sensor()
            sensor.key = ndb.Key('Sensor', sensor_name)
            sensor.put()
            sensor_names = sensor_name if sensor_names == "" else sensor_names +
                           "," + sensor_name
            memcache.set('sensor_names', sensor_names)
        memcache.set(sensor_name + '_value', sensor_value,
                     SENSOR_TIMEOUT_IN_SECONDS)
        return "Success\n"

# this function is accessible only using a http GET method, it returns a
# json object with the sensor's names list
@app.route('/sensor_names', methods=['GET'])
def print_names():
    return json.dumps([sensor.key.id() for sensor in Sensor.sensor_list()])

# this function is accessible only through GET http method. This GET request
# has to contain two parameters one is  "sensor", the sensor name, and the
# other one is "first_timestamp", the timestamp of the first  point that has
# to be plotted (this last parameter in not mandatory, if missed it is assumed
# equal to 0.
@app.route('/graphing_data', methods=['GET'])
def graphing_data():
    first_timestamp = request.args.get('first_timestamp')
    sensor_name = request.args.get('sensor')
    if first_timestamp is None:
        first_timestamp = 0
    else:
        first_timestamp = float(first_timestamp)
    points = DataPoint.points_for_sensor(ndb.Key('Sensor', sensor_name))
    epoch = datetime.datetime(1970, 1, 1)
```

```
143     res = []
        for point in points.iter():
145         timestamp = (point.date - epoch).total_seconds()
            if timestamp > first_timestamp:
147             _res = {"timestamp": timestamp, "value": point.value}
                res.append(_res)
149         else:
                break
151     res.reverse()
        return json.dumps(res)
153 
# this function is accessible through GET http method and deletes all the
155 # point of sensors
@app.route('/clear', methods=['GET'])
157 def clear_data():
        points = DataPoint.query()
159     for point in points:
            point.key.delete()
161     for sensor in Sensor.sensor_list():
            sensor.key.delete()
163     return "Success"

165 # this function is accessible through both GET and POST http methods and it
# is in charge to store the picture of the beating plot. This function is
167 # normally used with the POST method where the image is passed through
# "Image.jpg" field. Using a PC is also possible to update an image
169 # just using a browser, thank to the GET method implementation
@app.route('/picture/submit', methods=['GET', 'POST'])
171 def set_picture():
        if request.method == 'POST':
173 
            _file = request.files['Image.jpg']
175 
            if _file:
177             cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_image."
                                               + _file.filename.rsplit('.', 1)[1],
179                                            mode='w', content_type="image/jpeg")
                _file.save(cloud_file)
181             cloud_file.close()
                return "Success"
183     else:
            return '''
185         <!doctype html>
            <title>Upload microscope file V.1</title>
187         <h1>Upload microscope file</h1>
            <form method="POST"
189             action=""
                role="form"
191             enctype="multipart/form-data">
```

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

```
241             cloud_file.close()
        return "Success"
243
245 # this function is accessible through a GET http method and it returns the
    # microscope video stored inside the datastore
247 @app.route('/video/view', methods=['GET'])
    def view_video():
249     for _file in cloudstorage.listbucket(BUCKET_NAME):
            if "microscope_video" in _file.filename:
251             _file = cloudstorage.stat(_file.filename)
                logging.info(_file.filename)
253             logging.info(_file.content_type)
                cloud_file = cloudstorage.open(_file.filename, mode='r')
255             response = make_response(cloud_file.read())
                cloud_file.close()
257             response.mimetype = _file.content_type
                return response
259     return "No file found"

261 # this function is accessible through a POST http method and it is in
    # charge to store the electrovalve status inside the memcache. This http
263 # POST has to have the following parameters: "name", the name that
    # identifies the valve, "status", the status of electrovalve (on, off)
265 @app.route('/add/electrovalve', methods=['POST'])
    def add_electrovalve():
267     key = request.form.get('name')
        message = request.form.get('status')
269     if key and message:
            if message == 'on' or message == 'On' or message == 'ON':
271             MESSAGES[key] = True
                memcache.set(key, True)
273
            else:
275             MESSAGES[key] = False
                memcache.set(key, False)
277
           # ev.put()
279     return '%r' % memcache.get(key)

281 # this function accessible through a GET method returns the value of the
    # valve identified through <name> field in the URL
283 @app.route('/electrovalves/<name>', methods=['GET'])
    def get_electrovalve(name):
285     ev = memcache.get(name);
        if ev is None:
287         return ' %r ' % MESSAGES[name] or "%s not found!" % name
        else:
289         return ' %r ' % memcache.get(name)
```

```
291 @app.errorhandler(404)
    def page_not_found(e):
293     """Return a custom 404 error."""
        return 'Sorry, nothing at this URL.', 404
295


297

    if __name__ == "__main__":
299     app.run()
```

A.4 GLASSWARE

Listing A.17: MainService.java

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

import com.google.android.glass.timeline.LiveCard;
import com.google.android.glass.timeline.LiveCard.PublishMode;

import android.app.PendingIntent;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.os.Environment;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.IBinder;
import android.util.Log;
import android.view.animation.Animation;

import com.google.android.glass.timeline.LiveCard;
import com.googlecode.charts4j.AxisLabelsFactory;
import com.googlecode.charts4j.Data;
import com.googlecode.charts4j.GCharts;
import com.googlecode.charts4j.LineChart;
import com.googlecode.charts4j.Plot;
import com.googlecode.charts4j.Plots;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphViewSeries;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.json.JSONTokener;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.LineGraphView;


import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
```

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


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

    private Bitmap bmp;

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



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

    private AppDrawer mCallback;
```

```
94      private LiveCard mLiveCard;

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

102     /** INT associated to the Menu view request  */
        private static final int MENU = 0;
104     /** INT associated to the PH view request  */
        private static final int PH = 1;
106     /** INT associated to the Menu view request  */
        private static final int TEMPERATURE = 2;
108     /** INT associated to the Video view request  */
        private static final int VIDEO = 3;
110     /** INT associated to the Beating view request  */
        private static final int BEATING = 4;
112
        /** updating data period (in ms)  */
114     private static final long DATA_UPDATE_DELAY_MILLIS = 500;
        /** updating graph period (in ms)  */
116     private static final long GRAPH_UPDATE_DELAY_MILLIS = 500;
        /** updating video period (in ms)  */
118     private static final long FRAME_TIME_MILLIS = 100;
        private static final long VIDEO_UPDATE_DELAY_MILLIS = 60*1000; //time for video
120
        /** Runnable which describes the task to update the pH and Temperature
122     * sensors values */
        private final UpdateSensorValuesRunnable mUpdateSensorValuesRunnable =
124                                         new UpdateSensorValuesRunnable();
        /** Runnable which describes the task to compute the pH and Temperature
126     * graph (starting from their value)  */
        private final UpdateSensorGraphsRunnable mUpdateSensorGraphsRunnable =
128                                         new UpdateSensorGraphsRunnable();

130     private final UpdateMicroscopeVideoRunnable mUpdateMicroscopeVideoRunnable =
                                          new UpdateMicroscopeVideoRunnable();
132
       /** Runnable which describes the task to update the Electrovalves status */
134     private final UpdateElectrovalvesStatus mUpdateElectrovalvesStatus =
                                              new UpdateElectrovalvesStatus();
136


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

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

146 //          Hash table used for creating the graphs                    */
    /** Hash table in which the sensors values are stored  */
148 private Map<String,Double> mCurrentSensorValues;
    /** Hash table in which the graphs points are stored  */
150 private Map<String, ArrayList<DataPoint>> mSensorGraphData;
    /** Hash table in which the sensors graphs are stored */
152 private Map<String,Bitmap> mCurrentSensorGraphs;

154 private Map<String,Double> mSensorAverage;

156 @Override
    public IBinder onBind(Intent intent) {
158     return null;
    }
160
    @Override
162 public int onStartCommand(Intent intent, int flags, int startId) {
        if (mLiveCard == null) {
164         AppManager.getInstance().setState(MENU);
            mLiveCard = new LiveCard(this, LIVE_CARD_TAG);
166
            // Keep track of the callback to remove it before unpublishing.
168         mCallback = new AppDrawer(this);
            mLiveCard.setDirectRenderingEnabled(true).getSurfaceHolder().
170                                             addCallback(mCallback);

172         Intent menuIntent = new Intent(this, MenuActivity.class);
            menuIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
174                     Intent.FLAG_ACTIVITY_CLEAR_TASK);
            mLiveCard.setAction(PendingIntent.getActivity(this, 0, menuIntent, 0));
176         mLiveCard.attach(this);
            mLiveCard.publish(PublishMode.REVEAL);
178

180          /* Launch the task to update the data in another thread  */
            mHandlerThread = new HandlerThread("myHandlerThread");
182         mHandlerThread.start();
            mHandler = new Handler(mHandlerThread.getLooper());
184         DataTask task = new DataTask();
            task.execute(this);
186     } else {
            if(intent.getStringExtra(Intent.EXTRA_TEXT)!= null) {
188             Action action = Action.valueOf(intent.getStringExtra
                                        (Intent.EXTRA_TEXT));
190             switch (action) {
                    case Menu:
```

```
                        Log.i(LIVE_CARD_TAG, "State = Menu");
                        AppManager.getInstance().setState(MENU);
                        break;
                    case Beating:
                        Log.i(LIVE_CARD_TAG, "State = Beating");
                        AppManager.getInstance().setState(BEATING);
                        break;
                    case pH:
                        Log.i(LIVE_CARD_TAG, "State = pH");
                        AppManager.getInstance().setState(PH);
                        break;
                    case Temperature:
                        Log.i(LIVE_CARD_TAG, "State = Temperature");
                        AppManager.getInstance().setState(TEMPERATURE);
                        break;
                    case Video:
                        Log.i(LIVE_CARD_TAG, "State = Video");
                        AppManager.getInstance().setState(VIDEO);
                        break;
          case Electrovalves:
              Log.i(LIVE_CARD_TAG, "State = Electrovalves");
            AppManager.getInstance().setState(ELECTROVALVES);
            break;
                    default:
                        mLiveCard.navigate();
                        break;
                }
            }
            else
            {
                mLiveCard.navigate();
            }
        }

        // 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.
        return START_NOT_STICKY;
    }

    @Override
    public void onDestroy() {
        // Stop the Task which update the beating image
        if(!mImageDownloader.isStopped())
        {
            mImageDownloader.setIsStopped(true);
            Log.i(BEATING_TAG,"Removed Task");
        }
```

```
        // Stop the Task which update the sensors Graphs
242     if(!mUpdateSensorGraphsRunnable.isStopped())
        {
244         mUpdateSensorGraphsRunnable.setStop(true);
            Log.i(PH_TAG + " " + TEMPERATURE_TAG,"Removed Task of Graphs");
246     }
        // Stop the Task which update the sensors value
248     if(!mUpdateSensorValuesRunnable.isStopped())
        {
250         mUpdateSensorValuesRunnable.setStop(true);
            Log.i(PH_TAG + " " + TEMPERATURE_TAG,"Removed Task of Values");
252     }
        if (!mUpdateMicroscopeVideoRunnable.isStopped())
254     {
            mUpdateMicroscopeVideoRunnable.setStop(true);
256         Log.i(VIDEO_TAG, "Removed Task of Uploading values");
        }
258 if (!mUpdateElectrovalvesStatus.isStopped())
        {
260         mUpdateElectrovalvesStatus.setStop(true);
            Log.i(ELECTROVALVES_TAG, "Removed Task for getting
262                         Electrovalves status");
        }
264     if (mLiveCard != null && mLiveCard.isPublished()) {
            mLiveCard.unpublish();
266         mLiveCard = null;
        }
268     super.onDestroy();
    }
270
    /**
272  * Asynchronous Task for downloading the graphs and video in background
     */
274 private class DataTask extends AsyncTask<MainService,Void,Void>
    {
276
        @Override
278     protected Void doInBackground(MainService... params) {

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

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

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

```
290             {
                    mCurrentSensorValues.put(mSensor, 0.0);
292                 mSensorAverage.put(mSensor, 0.0);
                    mSensorGraphData.put(mSensor, new ArrayList<DataPoint>());
294                 mCurrentSensorGraphs.put(mSensor, null);
                }
296
                Log.i(LIVE_CARD_TAG, "Download the data");
298
                mUpdateSensorValuesRunnable.run();
300             // give the hash table to the Callback
                // mCallback.setSensorValues(mCurrentSensorValues);
302             // compute the graphs with previous values and give them to
            // the AppDrawer
304             mUpdateSensorGraphsRunnable.run();

306             // give the hash table to the Callback
                // mCallback.setSensorGraphs(mCurrentSensorGraphs);
308             // download the beating image and give this to the AppDrawer
                mImageDownloader.run();
310
                mUpdateMicroscopeVideoRunnable.run();
312
          mUpdateElectrovalvesStatus.run();
314             return null;
            }
316     }

318     /**
         * This runnable updates the sensors values taking them from the google engine
320      */
        private class UpdateSensorValuesRunnable implements Runnable
322     {
            private boolean mIsStopped = false;
324
            /** It implements the task of runnable
326          *
             * @see UpdateSensorValuesRunnable
328          */
            @Override
330         public void run()
            {
332             if (!isStopped())
                {
334                 /** JavaScript object, where the sensor values are
                  * temporary stored */
336                 JSONObject values = getSensorValues();
                    if (values != null)
338                 {
```

```
                    for (String mSensor : mSensors)
                    {
                        try
                        {
                            // update the hash table of sensor values
                            mCurrentSensorValues.put(mSensor,
                                values.getDouble(mSensor));

                        }
                        catch (JSONException ignored)
                        {}
                    }
                }

                // restart the Runnable after a given amount of time
                mHandler.postDelayed(mUpdateSensorValuesRunnable,
                                DATA_UPDATA_DELAY_MILLIS);
            }
        }



        /** It is the getter function that shows the status of runnable
         *
         * @return mIsStopped, true if the runnable is stopped, false otherwise
         */
        public boolean isStopped()
        {
            return mIsStopped;
        }

        /** It is the setter that allows to stop the runnable
         *
         * @param isStopped, true if the user wishes to stop the runnable,
     * false otherwise
         */
        public void setStop(boolean isStopped)
        {
            this.mIsStopped = isStopped;
        }
    }

    /** It gets the values of all of the sensors
     *
     * @return JSONOObject which contains all the values of sensors
     */
    private JSONObject getSensorValues()
    {
        try
```

```
388         {
                String values =
390             getURL("http://CENSURED/sensor_values");
                // return a JSONObject constructed by JSONTokener, which takes
392       // a source string and extracts characters and tokens from it
                return new JSONObject(new JSONTokener(values));
394         }
            catch (Exception e)
396         {
                Log.e(LIVE_CARD_TAG,"Failed to get sensor values",e);
398             return null;
            }
400     }


402
    // Get the new data points that we will need to graph
404
    /** This function computes the points that have to be plotted
406      *
      * @param sensor , the name of sensor (pH or Temperature)
408      * @param last_timestamp , the previous value of timestamp
      * @return JSONArray, return a list of values that corresponds to the
410   *   points that have to be plotted
      */
412     private JSONArray getDataPoints(String sensor, double last_timestamp)
    {
414         try
            {
416             String url =
            "http://CENSURED/graphing_data?sensor=" +
418          sensor + "&last_timestamp=" + String.valueOf(last_timestamp);
                String values = getURL(url);
420             return new JSONArray(new JSONTokener(values));
            }
422         catch (Exception e)
            {
424             Log.e(LIVE_CARD_TAG,"Failed to get data points",e);
                return null;
426         }
    }
428
    /** This method gets data from the given url website
430      *
      * @param _url, url in which the data are contained
432      * @return String, contained data from the given url
      * @throws IOException if an IO exception occurred during the download
434      */
    private String getURL(String _url) throws IOException
436 {
```

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

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

```
486                     if (curr_data.size() == 0) {
                            continue;
488                     }
                        // Store the timestamps and values in separate arrays to graph
490                     ArrayList<Double> timestamps = new ArrayList<Double>();
                        ArrayList<Double> values = new ArrayList<Double>();
492                     for (DataPoint curr_point : curr_data) {
                            timestamps.add(curr_point.getTimestamp());
494                         values.add(curr_point.getValue());
                        }
496
                        mSensorAverage.put(curr_sensor,computeAverage(values));
498                     // Scale the timestamp data
                        double maxTimestamp = Collections.max(timestamps);
500                     double minTimestamp = Collections.min(timestamps);
                        maxTimestamp *= 1.2;
502                     minTimestamp *= 0.8;
                        double intervalSize = maxTimestamp - minTimestamp;
504                     for (int i1 = 0; i1 < timestamps.size(); i1++) {
                            double currTimestamp = timestamps.get(i1);
506                         currTimestamp -= minTimestamp;
                            currTimestamp /= intervalSize;
508                         currTimestamp *= 100;
                            timestamps.set(i1, currTimestamp);
510                     }
                        // Scale the value data
512                     double maxVal = Collections.max(values);
                        double minVal = Collections.min(values);
514                     maxVal *= 1.2;
                        minVal *= 0.8;
516                     intervalSize = maxVal - minVal;
                        for (int i1 = 0; i1 < values.size(); i1++) {
518                         double currVal = values.get(i1);
                            currVal -= minVal;
520                         currVal /= intervalSize;
                            currVal *= 100;
522                         values.set(i1, currVal);

524                     }

526                     // Data xData = Data.newData(timestamps);
                        Data yData = Data.newData(values);
528                     Plot plot = Plots.newPlot(yData);
                        LineChart lineChart = GCharts.newLineChart(plot);
530                     lineChart.setSize(400, 200);
                        lineChart.addYAxisLabels(AxisLabelsFactory
532                     .newNumericRangeAxisLabels(minVal, maxVal));


534
```

```
                    mCurrentSensorGraphs.put(curr_sensor,
                     getBitmapFromURL(lineChart.toURLString()));

                }

                mCallback.SetSensorAvg(mSensorAverage);
                mCallback.setSensorGraphs(mCurrentSensorGraphs);
                Log.i("Main Service", "Graphs updated");

                // restart it after a given amount of time
                mHandler.postDelayed(mUpdateSensorGraphsRunnable,
                 GRAPH_UPDATE_DELAY_MILLIS);
            }
        }

        private double computeAverage(ArrayList<Double> values) {
            double sum = 0.0;
            if(!values.isEmpty()){
                for(Double value : values){
                    sum += value;
                }
                return sum/values.size();
            }
            return sum;
        }

        /** It is the getter function that shows the status of runnable
         *
         * @return mIsStopped, true if the runnable is stopped, false otherwise
         */
        public boolean isStopped()
        {
            return mIsStopped;
        }

        /** It is the setter that allows to stop the runnable
         *
         * @param isStopped, true if the user wishes to stop the runnable,
     *  false otherwise
         */
        public void setStop(boolean isStopped)
        {
            this.mIsStopped = isStopped;
        }
    }

    /** Runnable that implements the task for downloading beating image   */
    public class ImageDownloader implements Runnable {
        private String url;
```

```
584         private Context c;
            private boolean mIsStopped = false;
586
            /** Class constructor of ImageDownloader runnable
588          *
             * @param url, url link of image
590          * @param c, is the context in which the request of downloading image
        *        has been sent
592          */
            public ImageDownloader(String url,Context c )
594         {
                this.url = url;
596             this.c = c;
            }
598
            /** It implements the task of ImageDownloader runnable
600          *
             * @see
602          */
            @Override
604         public void run()
            {
606             if(!isStopped())
                {
608                 bmp = getBitmapFromURL(url);

610                 mCallback.setBMP(bmp);
                    Log.i(BEATING_TAG,"bmp settato");
612             }
            }
614
            /** It is the getter function that shows the status of ImageDownloader
616     * runnable
             *
618          * @return mIsStopped, true if the runnable is stopped, false otherwise
             */
620         public boolean isStopped()
            {
622             return mIsStopped;
            }
624
            /** It is the setter that allows to stop the runnable
626          *
             * @param isStopped, true if the user wishes to stop the runnable,
628          * false otherwise
             */
630         public void setIsStopped(boolean isStopped)
            {
632             this.mIsStopped = isStopped;
```

```
        }
    }
    /** This method pulls an image from the given url
     *
     * @param urlLink where the image is stored
     * @return Bitmap which contains the image of the graph
     * @throws java.io.IOException if an IO exception occurred during the download
     */
    public static Bitmap getBitmapFromURL(String urlLink){
        try{
            Log.i(BEATING_TAG, "start downloading image ");
            long startTime = System.currentTimeMillis();
            URL url = new URL(urlLink);
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setDoInput(true);
            connection.connect();
            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 (!isStopped()) {
                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() {
```

```
682         try {
                URL url = new URL("http://CENSURED/video/view");
684             long startTime = System.currentTimeMillis();
                Log.i(VIDEO_TAG, "video download beginning: "+url);
686             URLConnection ucon = url.openConnection();
                ucon.setReadTimeout(0);
688             ucon.setConnectTimeout(0);
                // Define InputStreams to read from the URLConnection.
690             InputStream is = ucon.getInputStream();
                BufferedInputStream inStream = new BufferedInputStream(is, 1024*5);
692             File file = new File(TEMP_VIDEO_FILE_NAME);

694             FileOutputStream outStream = new FileOutputStream(file);

696             FileLock lock = outStream.getChannel().lock();
                byte[] buff = new byte[1024*5];
698             // Read bytes (and store them) until there is nothing more to read(-1)
                int len;
700             while ((len = inStream.read(buff)) != -1) {
                    outStream.write(buff,0,len);
702             }
                // Clean up
704
                outStream.flush();
706             lock.release();
                outStream.close();
708             inStream.close();
                Log.i(VIDEO_TAG, "download completed in "
710                     + ((System.currentTimeMillis() - startTime) / 1000)
                        + " sec");
712         }
            catch (IOException e) {
714             Log.e(VIDEO_TAG, "Failed to download microscope video", e);
            }
716     }

718     private class UpdateElectrovalvesStatus implements Runnable {
            private boolean mIsStopped = false;
720
            public void run(){
722             if (!isStopped()){
                    try {
724                     getElectrovalvesStatus();
                    } catch (IOException e) {
726                     e.printStackTrace();
                    } catch (InterruptedException e) {
728                     e.printStackTrace();
                    }
730                 Log.i(LIVE_CARD_TAG, "Status Updated");
```

```
            }
        }
        public boolean isStopped(){
            return mIsStopped;
        }
        public void setStop(boolean isStopped){
            this.mIsStopped = isStopped;
        }
    }

    public static void getElectrovalvesStatus() throws IOException,
                                                    InterruptedException {
        for(int index = 0; index < Electrovalves.ELECTROVALVES_NUMBER; index++){
            String res = HTTPget(Electrovalves.URL_ELECTROVALVES_BASE + "EV" +
                    Integer.toString(index+1));
            if(res.equals(" True ")){
                Electrovalves.setElectrovalvesStatus(index,true);
            }
            else if(res.equals(" False ")) {
                Electrovalves.setElectrovalvesStatus(index,false);
            }
            Log.i("EV"+Integer.toString(index+1), res);

        }
    }


    private static String HTTPget(String link) throws IOException,
                                                    InterruptedException {
        URL url = new URL(link);
        HttpURLConnection mUrlConnection = (HttpURLConnection) url.openConnection();
        InputStream in = new BufferedInputStream(mUrlConnection.getInputStream());
        int ch;
        StringBuffer b = new StringBuffer();
        while ((ch = in.read()) != -1){
            b.append((char) ch);
        }

        String mResult = new String(b);
        return mResult;

    }

    // convert inputstream to String
    private static String convertInputStreamToString(InputStream inputStream)
                                                        throws IOException{
        BufferedReader bufferedReader = new BufferedReader(
                                    new InputStreamReader(inputStream));
        String line = "";
```

```
780         String result = "";
            while((line = bufferedReader.readLine()) != null)
782             result += line;

784         inputStream.close();
            return result;
786
    }
788


790 }
```

Listing A.18: AppDrawer.java

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

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

18 import java.util.Map;

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

```
36  * required to properly measure and layout the {@link View}'s dimension.
    */
38 public class AppDrawer. implements DirectRenderingCallback {
       /** INT associated to the Menu view request  */
40     private static final int MENU = 0;
       /** INT associated to the PH view request  */
42     private static final int PH = 1;
       /** INT associated to the Menu view request  */
44     private static final int TEMPERATURE = 2;
       /** INT associated to the Video view request  */
46     private static final int VIDEO = 3;
       /** INT associated to the Beating view request  */
48     private static final int BEATING = 4;

50     /** Bitmap in which the beating image is stored */
       private Bitmap bmp;
52     private boolean mReady;

54     private static final String TAG = AppDrawer..class.getSimpleName();
     /** View object of the GlassWear Main window */
56     private final MainView mMainView;
     /** View object of the GlassWear Beating window */
58     private final BeatingView mBeatingView;
       /** View object of the GlassWear pH window */
60     private final PHViewer mPhViewer;
     /** View object of the GlassWear Temperature window */
62     private final TemperatureView mTemperatureView;

64     private SurfaceHolder mHolder;
       private boolean mRenderingPaused;
66
       /** Hash table in which the sensors graphs are stored */
68     private Map<String,Bitmap> mCurrentSensorGraphs;
       /** Hash table in which the sensors values are stored  */
70     private Map<String,Double> mCurrentSensorValues;
     /** Hash table in which the sensors average values are stored  */
72     private Map<String,Double> mSensorAverage;
       private GraphView mGraphView;
74
       private final MainView.Listener mMainListener = new MainView.Listener() {
76


78         @Override
           public void onChange() {
80             updateRenderingState();
           }
82     };

84     /** Defines the Listener of pH viewer, it is used to communicate with
```

```
     * that viewer and allowing its viewing */
86   private final PHViewer.Listener mPhListener = new PHViewer.Listener(){
         @Override
88       public void onChange(){
                 updateRenderingState();
90       }
     };
92
   /** Defines the Listener of Beating viewer, it is used to communicate with
94     * that viewer and allowing its viewing */
     private final BeatingView.Listener mBeatingListener =
96                                   new BeatingView.Listener() {

98       @Override
         public void onChange() {
100          updateRenderingState();
         }
102  };

104  /** Defines the Listener of Temperature viewer, it is used to communicate with
       * that viewer and allowing its viewing */
106    private final TemperatureView.Listener mTemperatureListener =
                                     new TemperatureView.Listener() {
108
         @Override
110      public void onChange() {
             updateRenderingState();
112      }
     };
114
     /** Defines the ManageBitmap of Beating viewer, it is used to communicate with
116    * that viewer in order to update the bitmap to be displayed*/
     private final BeatingView.ManageBitmap mBeatingBitmap =
118                                        new BeatingView.ManageBitmap(){
         @Override
120      public Bitmap getBitmap() {
             return bmp;
122      }

124      public boolean isReady(){
             return mReady;
126      }
     };
128
     /** Defines the ManageDataGraph of PH viewer, it is used to communicate with
130    * that viewer in order to update the hash map which contains the graph to
   * be displayed*/
132    private final PHViewer.ManageBitmap mPHManageDataGraph =
                                       new PHViewer.ManageBitmap(){
```

```
134
        public Bitmap getBitmap() {
136         return mCurrentSensorGraphs.get("pH");
        }
138
        public double getAvg(){return mSensorAverage.get("pH");}
140
        public boolean isReady(){
142         return mReady;
        }
144  };

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

152     public Bitmap getBitmap() {
            return mCurrentSensorGraphs.get("pH");
154     }

156     public double getAvg(){return mSensorAverage.get("Temperature");}

158     public boolean isReady(){
            return mReady;
160     }
     };
162


164  public AppDrawer.(Context context) {
        this(new MainView(context), new BeatingView(context),
166      new PHViewer(context), new TemperatureView(context));
     }
168
     public AppDrawer.(MainView countDownView, BeatingView chronometerView,
170                  PHViewer phViewer, TemperatureView temperatureView) {
        mMainView = countDownView;
172     mMainView.setListener(mMainListener);

174     mBeatingView = chronometerView;
        mBeatingView.setListener(mBeatingListener);
176     mBeatingView.setManageBPM(mBeatingBitmap);

178     mPhViewer = phViewer;
        mPhViewer.setListener(mPhListener);
180     mPhViewer.setManageBPM(mPHManageDataGraph);

182     mTemperatureView = temperatureView;
```

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

        mReady = false;
    }

    /**
     * Uses the provided the width and height to measure and layout the
     * inflated View.
     */
    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
                               int height) {
        // Measure and layout the view with the canvas dimensions.
        int measuredWidth = View.MeasureSpec.makeMeasureSpec(width,
                        View.MeasureSpec.EXACTLY);
        int measuredHeight = View.MeasureSpec.makeMeasureSpec(height,
                        View.MeasureSpec.EXACTLY);

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

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

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

        mTemperatureView.measure(measuredWidth, measuredHeight);
        mTemperatureView.layout(0, 0, mTemperatureView.getMeasuredWidth(),
                            mTemperatureView.getMeasuredHeight());
    }

    /**
     * Keeps the created SurfaceHolder and updates this class' rendering state.
     */
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        // The creation of a new Surface implicitly resumes the rendering.
        mRenderingPaused = false;
        mHolder = holder;
        updateRenderingState();
    }

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

```
232      */
         @Override
234      public void surfaceDestroyed(SurfaceHolder holder) {
             mHolder = null;
236          updateRenderingState();
         }
238
         /**
240       * Updates this class rendering state according to the provided paused flag.
          */
242      @Override
         public void renderingPaused(SurfaceHolder holder, boolean paused) {
244          mRenderingPaused = paused;
             updateRenderingState();
246      }

248      /**
          * Starts or stops rendering according to the LiveCard's state.
250       */
         private void updateRenderingState() {
252          if (mHolder != null && !mRenderingPaused) {
                 switch (AppManager.getInstance().getState())
254              {
                     case MENU:
256                      draw(mMainView);
                         mBeatingView.stop();
258                      mPhViewer.stop();
                         mTemperatureView.stop();
260                      mMainView.start();
                         break;
262                  case BEATING:
                         draw(mBeatingView);
264                      mMainView.stop();
                         mPhViewer.stop();
266                      mTemperatureView.stop();
                         mBeatingView.start();
268                      break;
                     case PH:
270                      draw(mPhViewer);
                         mMainView.stop();
272                      mBeatingView.stop();
                         mTemperatureView.stop();
274                      mPhViewer.start();
                         break;
276                  case TEMPERATURE:
                         draw(mTemperatureView);
278                      mMainView.stop();
                         mBeatingView.stop();
280                      mPhViewer.stop();
```

```
                    mTemperatureView.start();
                    break;
                case VIDEO:
                    mTemperatureView.stop();
                    mBeatingView.stop();
                    mMainView.stop();
                    mPhViewer.stop();
                    break;
                default:
                    draw(mMainView);
                    mPhViewer.stop();
                    mBeatingView.stop();
                    mTemperatureView.stop();
                    mMainView.start();
                    break;

            }

        } else {
            mMainView.stop();
            mBeatingView.stop();
            mPhViewer.stop();
            mTemperatureView.stop();
        }

    }

    /**
     * Draws the view in the SurfaceHolder's canvas.
     */
    private void draw(View view) {

        Canvas canvas;

        try {
            canvas = mHolder.lockCanvas();
        } catch (Exception e) {
            Log.e(TAG, "Unable to lock canvas: " + e);
            return;
        }
        if (canvas != null) {

            view.draw(canvas);
            mHolder.unlockCanvasAndPost(canvas);
        }
    }
    /** Setter for the beating graph
     *
     * @param bmp ,  Bitmap which contains the beating graph
```

```
330      */
         public void setBMP(Bitmap bmp){
332
             this.bmp = bmp;
334          this.mReady = true;
             Log.i("DRAWER", "bmp settato");
336      }

338  /** Setter for the sensors Graphs hash map
         *
340      * @param sensorGraphs ,  Map<String,Double> sensorGraphs
     *                            which contains the hash
342      *                            map with graphs of the sensors
         */
344      public void setSensorGraphs(  Map<String,Bitmap> sensorGraphs ){
             this.mCurrentSensorGraphs = sensorGraphs;
346      }

348  /** Setter for the sensors viewer hash map
         *
350      * @param graphView ,  Map<String,Double> sensorGraphs
     *                            which contains the hash
352      *                            map with viewer of the sensors
         */
354      public void setPHGraphViewer(  GraphView graphView ){
             this.mGraphView = graphView;
356      }

358      /** Setter for the sensors value hash map
         *
360      * @param currentSensorValues ,  Map<String,Double> currentSensorValues
     *                                which contains the hash
362      *                                map with the current value of the sensors
         */
364      public void setSensorValues(  Map<String,Double> currentSensorValues ){
             this.mCurrentSensorValues = currentSensorValues;
366      }

368      /** Setter for the sensors average values hash map
         *
370      * @param sensorAvg ,  Map<String,Double> currentSensorValues
     *                                which contains the hash
372      *                                map with the average values of the sensors
         */
374      public  void SetSensorAvg (Map<String,Double> sensorAvg){
             this.mSensorAverage = sensorAvg;
376      }


378
```

```
380 }
```

Listing A.19: MainView.java

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

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

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

public class MainView extends FrameLayout {

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

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

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

    private final TextView timeText;

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

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

```
    };

    private Listener mListener;
    private boolean mRunning = false;

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

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

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

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

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

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

    /**
     * Starts the countdown animation if not yet started.
     */
    public void start() {
        if (!mRunning) {
            postDelayed(mUpdateViewRunnable, 0);
        }
        mRunning = true;
    }

    /**
```

```
94      * Stops the chronometer.
        */
96     public void stop() {
           if (mRunning) {
98             removeCallbacks(mUpdateViewRunnable);
              // mStarted = false;
100        }
           mRunning = false;
102    }

104    void updateView() {
           if (mRunning) {
106      timeText.setText( new SimpleDateFormat("hh:mm a")
                 .format(new Date()));
108            if (mListener != null) {
                   mListener.onChange();
110            }

112        }
       }
114 }
```

Listing A.20: BeatingView.java

```
   package com.google.android.glass.sample.klabinterface;
2

   import android.content.Context;
4  import android.graphics.Bitmap;
   import android.graphics.drawable.Drawable;
6  import android.os.Handler;
   import android.text.Layout;
8  import android.util.AttributeSet;
   import android.util.Log;
10 import android.view.LayoutInflater;
   import android.widget.FrameLayout;
12 import android.widget.ImageView;
   import android.widget.TextView;
14

   public class BeatingView extends FrameLayout {
16

       boolean mState = false;
18     private Bitmap bmp;

20     /**
        * Interface to listen for changes on the view layout.
22      */
       public interface Listener {
24         /** Notified of a change in the view. */
           public void onChange();
```

```
26      }

28      /** About 24 FPS, visible for testing. */
        static final long DELAY_MILLIS = 41;
30
        private ImageView beatingImage;
32      private final TextView mTitle;

34      private final Handler mHandler = new Handler();
        private final Runnable mUpdateTextRunnable = new Runnable() {
36
            @Override
38          public void run() {
                if (mRunning) {
40                  updateText();
                    postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
42              }
            }
44      };

46      private boolean mRunning;

48      public interface ManageBitmap{
            public Bitmap getBitmap();
50
            public boolean isReady();
52      }

54      private Listener mChangeListener;

56      private ManageBitmap mManage;

58      public BeatingView(Context context) {
            this(context, null, 0);
60      }

62      public BeatingView(Context context, AttributeSet attrs) {
            this(context, attrs, 0);
64      }

66      public BeatingView(Context context, AttributeSet attrs, int style) {
            super(context, attrs, style);
68          LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
            mTitle = (TextView) findViewById(R.id.message);
70          beatingImage = (ImageView) findViewById(R.id.image_left);
            mTitle.setText("Beating");
72          int id = getResources().getIdentifier(
                  "com.google.android.glass.sample.stopwatch:drawable/layout_color",
74                null,null);
```

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

78  /**
     * Sets a Listener
80   */
    public void setListener(Listener listener) {
82      mChangeListener = listener;
    }
84

    /**
86   * Returns the set Listener.
     */
88  public Listener getListener() {
        return mChangeListener;
90  }

92  /**
     * Starts the BeatingView.
94   */
    public void start() {
96      if (!mRunning) {
            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
98      }
        mRunning = true;
100 }

102 /**
     * Stops the BeatingView.
104  */
    public void stop() {
106     if (mRunning) {
            removeCallbacks(mUpdateTextRunnable);
108     }
        mRunning = false;
110 }

112 @Override
    public boolean postDelayed(Runnable action, long delayMillis) {
114     return mHandler.postDelayed(action, delayMillis);
    }
116

    @Override
118 public boolean removeCallbacks(Runnable action) {
        mHandler.removeCallbacks(action);
120     return true;
    }
122 /**
     * Sets a {@link Listener}.
```

```
124      */
         public void setManageBPM(ManageBitmap manager) {
126          mManage = manager;
         }
128


130      /**
          * Updates the value of the chronometer, visible for testing.
132       */
         void updateText() {
134          if(mManage.isReady())
             {
136              Log.i("viewer", "ce prova");
                 bmp = mManage.getBitmap();
138              beatingImage.setImageBitmap(bmp);
             }
140          if (mChangeListener != null) {
                 mChangeListener.onChange();
142          }
         }
144  }
```

Listing A.21: PHViewer.java

```
   package com.google.android.glass.sample.klabinterface;
2

   import android.content.Context;
4  import android.graphics.Bitmap;
   import android.graphics.drawable.Drawable;
6  import android.os.Handler;
   import android.text.Layout;
8  import android.util.AttributeSet;
   import android.util.Log;
10 import android.view.LayoutInflater;
   import android.widget.FrameLayout;
12 import android.widget.ImageView;
   import android.widget.LinearLayout;
14 import android.widget.TextView;

16 import com.jjoe64.graphview.GraphView;
   import com.jjoe64.graphview.GraphViewSeries;
18 import com.jjoe64.graphview.LineGraphView;

20 public class PHViewer extends FrameLayout {

22     private Context context;
       private Bitmap bmp;
24

       /**
```

```
26      * Interface to listen for changes on the view layout.
        */
28      public interface Listener {
            /** Notified of a change in the view. */
30          public void onChange();
        }
32
        static final long DELAY_MILLIS = 41;
34
        private ImageView beatingImage;
36      private final TextView mTextTitle;
        private TextView AvgView;
38
        private final Handler mHandler = new Handler();
40      private final Runnable mUpdateTextRunnable = new Runnable() {

42          @Override
            public void run() {
44              if (mRunning) {
                    updateText();
46                  postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
                }
48          }
        };
50
        private boolean mRunning;
52
        public interface ManageBitmap{
54          public Bitmap getBitmap();

56          public boolean isReady();

58          public double getAvg();
        }
60      private Listener mChangeListener;

62      private ManageBitmap mManage;

64      public PHViewer(Context context) {
            this(context, null, 0);
66      }

68      public PHViewer(Context context, AttributeSet attrs) {
            this(context, attrs, 0);
70      }

72      public PHViewer(Context context, AttributeSet attrs, int style) {
            super(context, attrs, style);
74          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/layout_color"
            ,null,null);
        beatingImage.setImageResource(id);
    }

    /**
     * Sets a Listener.
     */
    public void setListener(Listener listener) {
        mChangeListener = listener;
    }

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

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

    /**
     * Stops the chronometer.
     */
    public void stop() {
        if (mRunning) {
            removeCallbacks(mUpdateTextRunnable);
        }
        mRunning = false;
    }

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

```
124     @Override
        public boolean removeCallbacks(Runnable action) {
126         mHandler.removeCallbacks(action);
            return true;
128     }
        /**
130      * Sets a Listener.
         */
132     public void setManageBPM(ManageBitmap manager) {
            mManage = manager;
134     }


136

        /**
138      * Updates the value of the chronometer, visible for testing.
         */
140     void updateText() {
            if(mManage.isReady())
142         {
                bmp = mManage.getBitmap();
144             AvgView.setText(Double.toString(mManage.getAvg()));
                beatingImage.setImageBitmap(bmp);
146         }
            if (mChangeListener != null) {
148             mChangeListener.onChange();
            }
150     }
}
```

Listing A.22: TemperatureView.java

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

3           import android.content.Context;
            import android.graphics.Bitmap;
5           import android.graphics.drawable.Drawable;
            import android.os.Handler;
7           import android.text.Layout;
            import android.util.AttributeSet;
9           import android.util.Log;
            import android.view.LayoutInflater;
11          import android.widget.FrameLayout;
            import android.widget.ImageView;
13          import android.widget.TextView;


15
    public class TemperatureView extends FrameLayout {
17

        private Bitmap bmp;
```

```
19      private TextView AvgView;

21      /**
         * Interface to listen for changes on the view layout.
23       */
        public interface Listener {
25          /** Notified of a change in the view. */
            public void onChange();
27      }

29      /** About 24 FPS, visible for testing. */
        static final long DELAY_MILLIS = 41;
31
        private ImageView beatingImage;
33      private final TextView mTitle;

35      private final Handler mHandler = new Handler();
        private final Runnable mUpdateTextRunnable = new Runnable() {
37
            @Override
39          public void run() {
                if (mRunning) {
41                  updateText();
                    postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
43              }
            }
45      };

47      private boolean mRunning;

49      public interface ManageBitmap{
            public Bitmap getBitmap();
51
            public boolean isReady();
53
            public double getAvg();
55      }
        private Listener mChangeListener;
57
        private ManageBitmap mManage;
59
        public TemperatureView(Context context) {
61          this(context, null, 0);
        }
63
        public TemperatureView(Context context, AttributeSet attrs) {
65          this(context, attrs, 0);
        }
67
```

```
    public TemperatureView(Context context, AttributeSet attrs, int style) {
69      super(context, attrs, style);
        LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
71      beatingImage = (ImageView) findViewById(R.id.image_left);
        mTitle = (TextView) findViewById(R.id.message);
73      AvgView = (TextView) findViewById(R.id.avg);

75      mTitle.setText("Temperature");
        int id = getResources().getIdentifier(
77           "com.google.android.glass.sample.stopwatch:drawable/layout_color"
         ,null,null);
79      beatingImage.setImageResource(id);
    }
81  /**
     * Sets a Listener.
83   */
    public void setListener(Listener listener) {
85      mChangeListener = listener;
    }
87

    /**
89   * Returns the set Listener.
     */
91  public Listener getListener() {
        return mChangeListener;
93  }

95  /**
     * Starts the chronometer.
97   */
    public void start() {
99      if (!mRunning) {
            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
101     }
        mRunning = true;
103 }

105 /**
     * Stops the chronometer.
107  */
    public void stop() {
109     if (mRunning) {
            removeCallbacks(mUpdateTextRunnable);
111     }
        mRunning = false;
113 }

115 @Override
    public boolean postDelayed(Runnable action, long delayMillis) {
```

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

    @Override
    public boolean removeCallbacks(Runnable action) {
        mHandler.removeCallbacks(action);
        return true;
    }
    /**
     * Sets a Listener.
     */
    public void setManageBPM(ManageBitmap manager) {
        mManage = manager;
    }

    void updateText() {
        if(mManage.isReady())
        {
            bmp = mManage.getBitmap();
            AvgView.setText(Double.toString(mManage.getAvg()));
            beatingImage.setImageBitmap(bmp);

        }
        if (mChangeListener != null) {
            mChangeListener.onChange();
        }
    }
}
```

Listing A.23: VideoPlayerActivity.java

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

import android.app.Activity;
import android.content.Intent;
import android.content.res.AssetManager;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.MediaController;
import android.widget.VideoView;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.io.OutputStream;
public class VideoPlayerActivity extends Activity {
   private static final int VIDEO_PLAY_REQUEST_CODE = 200;
   private static final String TAG = "VIDEO_TAG";
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       String filepath;
       Bundle extras = getIntent().getExtras();
       if (extras != null)
           filepath = extras.getString("filepath");
       else {
           filepath = copyAsset(VIDEO_FILE_NAME);
       }

       Intent i = new Intent();
       i.setAction("com.google.glass.action.VIDEOPLAYER");
       i.putExtra("video_url", filepath);
       startActivityForResult(i, VIDEO_PLAY_REQUEST_CODE);
   }
   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
       if (requestCode == VIDEO_PLAY_REQUEST_CODE )
           finish();
   }
   String copyAsset(String filename) {
       final String PATH = Environment.getExternalStorageDirectory().toString()
                           + "/myvideoapps/";
       File dir = new File(PATH);
       if (!dir.exists()) {
           if (!dir.mkdirs()) {
               Log.v(TAG, "ERROR: Creation of directory " + PATH
                + " on sdcard failed");
               return null;
           } else {
               Log.v(TAG, "Created directory " + PATH + " on sdcard");
           }
       }
       if (!(new File( PATH + filename).exists())) {
           try {
               AssetManager assetManager = getAssets();
               InputStream in = assetManager.open(filename);
               OutputStream out = new FileOutputStream(PATH + filename);
               byte[] buf = new byte[1024];
               int len;
               while ((len = in.read(buf)) > 0) {
                   out.write(buf, 0, len);
               }
               in.close();
               out.close();
           } catch (IOException e) {
```

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

Listing A.24: MenuActivity.java

```
package com.google.android.glass.sample.klabinterface;
2
import android.app.Activity;
4 import android.content.Intent;
import android.os.Bundle;
6 import android.os.Handler;
import android.view.Menu;
8 import android.view.MenuInflater;
import android.view.MenuItem;
10 import android.view.View;
import android.widget.Toast;
12
import java.lang.Runnable;
14
/**
16  * Activity showing the stopwatch options menu.
 */
18 public class MenuActivity extends Activity {

20     /** INT associated to the Menu view request  */
       private static final int MENU = 0;
22     /** INT associated to the PH view request  */
       private static final int PH = 1;
24     /** INT associated to the Menu view request  */
       private static final int TEMPERATURE = 2;
26     /** INT associated to the Video view request  */
       private static final int VIDEO = 3;
28     /** INT associated to the Beating view request  */
       private static final int BEATING = 4;
30   /** INT associated to the Electrovalves view request  */
       private static final int ELECTROVALVES = 5;
32
       private final Handler mHandler = new Handler();
34     private int state = 0;

36     @Override
       public void onAttachedToWindow() {
38         super.onAttachedToWindow();
           openOptionsMenu();
```

```
40      }
42      @Override
        public boolean onCreateOptionsMenu(Menu menu) {
44          MenuInflater inflater = getMenuInflater();
            inflater.inflate(R.menu.stopwatch, menu);
46          return true;
        }
48
        @Override
50      public boolean onPreparePanel(int featureId, View view, Menu menu) {
            AppManager appManager = AppManager.getInstance();
52
            boolean initialView = appManager.getState() == MENU;
54          boolean imageView = appManager.getState() == PH ||
                    appManager.getState() == TEMPERATURE ||
56                  appManager.getState() == BEATING ||
            appManager.getState() == ELECTROVALVES ||
58                  appManager.getState() == VIDEO;

60          setOptionsMenuState(menu, R.id.action_back, imageView);
            setOptionsMenuState(menu, R.id.action_view_ph, initialView);
62          setOptionsMenuState(menu, R.id.action_view_temperature, initialView);
            setOptionsMenuState(menu, R.id.action_view_video, initialView);
64          setOptionsMenuState(menu, R.id.action_view_beating, initialView);
            setOptionsMenuState(menu, R.id.action_drive_electrovalves, initialView);
66          setOptionsMenuState(menu, R.id.action_toggle_EV1, electrovalveView);
            setOptionsMenuState(menu, R.id.action_toggle_EV2, electrovalveView);
68          setOptionsMenuState(menu, R.id.action_toggle_EV3, electrovalveView);
            setOptionsMenuState(menu, R.id.action_toggle_EV4, electrovalveView);
70          setOptionsMenuState(menu, R.id.action_toggle_EV5, electrovalveView);
            setOptionsMenuState(menu, R.id.action_toggle_EV6, electrovalveView);
72          setOptionsMenuState(menu, R.id.action_toggle_EV7, electrovalveView);
            setOptionsMenuState(menu, R.id.action_toggle_EV8, electrovalveView);
74          setOptionsMenuState(menu, R.id.action_stop, true);

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

```
                    public void run() {
90                      stopService(new Intent(MenuActivity.this,
                        StopwatchService.class));
92                  }
                });
94              return true;
            case R.id.action_view_beating:
96              handleViewBeating();
                return true;
98          case R.id.action_back:
                handleViewBack();
100             return true;
            case R.id.action_view_temperature:
102             handleViewTemperature();
                return true;
104         case R.id.action_view_ph:
                handleViewPh();
106             return true;
            case R.id.action_view_video:
108             handleViewVideo();
                return true;
110     case R.id.action_drive_electrovalves:
                handleViewElectrovalves();
112             return true;
            case R.id.action_toggle_EV1:
114             handleToggleEV1();
                return true;
116         case R.id.action_toggle_EV2:
                handleToggleEV2();
118             return true;
            case R.id.action_toggle_EV3:
120             handleToggleEV3();
                return true;
122         case R.id.action_toggle_EV4:
                handleToggleEV4();
124             return true;
            case R.id.action_toggle_EV5:
126             handleToggleEV5();
                return true;
128         case R.id.action_toggle_EV6:
                handleToggleEV6();
130             return true;
            case R.id.action_toggle_EV7:
132             handleToggleEV7();
                return true;
134         case R.id.action_toggle_EV8:
                handleToggleEV8();
136             return true;
            default:
```

```
138                 return super.onOptionsItemSelected(item);
            }
140     }

142     private void handleViewVideo() {
            Toast.makeText(this, "Video", Toast.LENGTH_SHORT).show();
144         // launch a new thread for starting a new live card
            mHandler.post(new Runnable() {
146             @Override
                public void run() {
148                 Intent intent = new Intent(MenuActivity.this,
                            StopwatchService.class);
150                 intent.putExtra(Intent.EXTRA_TEXT, "Video");
                    startService(intent);
152             }
            });
154     }

156     private void handleViewTemperature() {
            Toast.makeText(this, "Temperature", Toast.LENGTH_SHORT).show();
158         // launch a new thread for starting a new live card
            mHandler.post(new Runnable() {
160             @Override
                public void run() {
162                 Intent intent = new Intent(MenuActivity.this,
                            StopwatchService.class);
164                 intent.putExtra(Intent.EXTRA_TEXT, "Temperature");
                    startService(intent);
166             }
            });
168     }

170     private void handleViewPh() {
            Toast.makeText(this, "Ph", Toast.LENGTH_SHORT).show();
172         // launch a new thread for starting a new live card
            mHandler.post(new Runnable() {
174             @Override
                public void run() {
176                 Intent intent = new Intent(MenuActivity.this,
                            StopwatchService.class);
178                 intent.putExtra(Intent.EXTRA_TEXT, "pH");
                    startService(intent);
180             }
            });
182     }

184     private void handleViewBack() {
            Toast.makeText(this, "Menu", Toast.LENGTH_SHORT).show();
186         // launch a new thread for starting a new live card
```

```
        mHandler.post(new Runnable() {
188         @Override
            public void run() {
190             Intent intent = new Intent(MenuActivity.this,
                        StopwatchService.class);
192             intent.putExtra(Intent.EXTRA_TEXT, "Menu");
                startService(intent);
194         }
        });
196 }

198 @Override
    public void onOptionsMenuClosed(Menu menu) {
200     // Nothing else to do, closing the Activity.
        finish();
202 }

204 /**
     * Posts a Runnable at the end of the message loop, overridable for testing.
206  */
    protected void post(Runnable runnable) {
208     mHandler.post(runnable);
    }
210

    /**
212  * The function handle the request to view the beating plot
     */
214 private void handleViewBeating() {
        Toast.makeText(this, "Beating", Toast.LENGTH_SHORT).show();
216     // launch a new thread for starting a new live card
        mHandler.post(new Runnable() {
218         @Override
            public void run() {
220             Intent intent = new Intent(MenuActivity.this,
                        StopwatchService.class);
222             intent.putExtra(Intent.EXTRA_TEXT, "Beating");
                startService(intent);
224         }
        });
226 }

228 /**
     * The function handle the request to toggle the electrovalve number 8
230  */
  private void handleToggleEV8()  {
232     Toast.makeText(this, "Toggle EV8", Toast.LENGTH_SHORT).show();
        mHandler.post(new Runnable() {
234         @Override
            public void run() {
```

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

254     }

256     /**
         * The function handle the request to toggle the electrovalve number 7
258      */
        private void handleToggleEV7() {
            Toast.makeText(this, "Toggle EV7", Toast.LENGTH_SHORT).show();
            mHandler.post(new Runnable() {
262             @Override
                public void run() {
                    Intent intent = new Intent(MenuActivity.this, HTTPService.class);
                    intent.putExtra("METHOD", "POST");
266                 if(Electrovalves.getElectrovalveStatus(6)) {
                        intent.putExtra("LINK",
268                             Electrovalves.URL_ELECTROVALVES_POST_BASE);
                        intent.putExtra("PARAM", "name=EV7&status=off");
270                     Electrovalves.setElectrovalvesStatus(6, false);
                    }
272                 else {
                        intent.putExtra("LINK",
274                             Electrovalves.URL_ELECTROVALVES_POST_BASE);
                        intent.putExtra("PARAM", "name=EV7&status=on");
276                     Electrovalves.setElectrovalvesStatus(6, true);
                    }
278                 startService(intent);
                }
280         });
        }
282
    /**
284      * The function handle the request to toggle the electrovalve number 6
```

```
     */
    private void handleToggleEV6() {
        Toast.makeText(this, "Toggle EV6", Toast.LENGTH_SHORT).show();
        mHandler.post(new Runnable() {
            @Override
            public void run() {
                Intent intent = new Intent(MenuActivity.this, HTTPService.class);
                intent.putExtra("METHOD","POST");
                if(Electrovalves.getElectrovalveStatus(5)) {
                    intent.putExtra("LINK",
                            Electrovalves.URL_ELECTROVALVES_POST_BASE);
                    intent.putExtra("PARAM", "name=EV6&status=off");
                    Electrovalves.setElectrovalvesStatus(5, false);
                }
                else {
                    intent.putExtra("LINK",
                            Electrovalves.URL_ELECTROVALVES_POST_BASE);
                    intent.putExtra("PARAM", "name=EV6&status=on");
                    Electrovalves.setElectrovalvesStatus(5, true);
                }
                startService(intent);
            }
        });
    }

  /**
     * The function handle the request to toggle the electrovalve number 5
     */
    private void handleToggleEV5() {
        Toast.makeText(this, "Toggle EV5", Toast.LENGTH_SHORT).show();
        mHandler.post(new Runnable() {
            @Override
            public void run() {
                Intent intent = new Intent(MenuActivity.this, HTTPService.class);
                intent.putExtra("METHOD","POST");
                if(Electrovalves.getElectrovalveStatus(4)) {
                    intent.putExtra("LINK",
                            Electrovalves.URL_ELECTROVALVES_POST_BASE);
                    intent.putExtra("PARAM", "name=EV5&status=off");
                    Electrovalves.setElectrovalvesStatus(4, false);
                }
                else {
                    intent.putExtra("LINK",
                            Electrovalves.URL_ELECTROVALVES_POST_BASE);
                    intent.putExtra("PARAM", "name=EV5&status=on");
                    Electrovalves.setElectrovalvesStatus(4, true);
                }
                startService(intent);
            }
```

```
334         });
    }
336
    /**
338      * The function handle the request to toggle the electrovalve number 4
     */
340     private void handleToggleEV4() {
        Toast.makeText(this, "Toggle EV4", Toast.LENGTH_SHORT).show();
342         mHandler.post(new Runnable() {
            @Override
344             public void run() {
                Intent intent = new Intent(MenuActivity.this, HTTPService.class);
346                 intent.putExtra("METHOD", "POST");
                if(Electrovalves.getElectrovalveStatus(3)) {
348                     intent.putExtra("LINK",
                            Electrovalves.URL_ELECTROVALVES_POST_BASE);
350                     intent.putExtra("PARAM", "name=EV4&status=off");
                    Electrovalves.setElectrovalvesStatus(3, false);
352                 }
                else {
354                     intent.putExtra("LINK",
                            Electrovalves.URL_ELECTROVALVES_POST_BASE);
356                     intent.putExtra("PARAM", "name=EV4&status=on");
                    Electrovalves.setElectrovalvesStatus(3, true);
358                 }
                startService(intent);
360             }
        });
362     }

364     /**
     * The function handle the request to toggle the electrovalve number 3
366      */
    private void handleToggleEV3() {
368         Toast.makeText(this, "Toggle EV3", Toast.LENGTH_SHORT).show();
        mHandler.post(new Runnable() {
370             @Override
            public void run() {
372                 Intent intent = new Intent(MenuActivity.this, HTTPService.class);
                intent.putExtra("METHOD", "POST");
374                 if(Electrovalves.getElectrovalveStatus(2)) {
                    intent.putExtra("LINK",
376                             Electrovalves.URL_ELECTROVALVES_POST_BASE);
                    intent.putExtra("PARAM", "name=EV3&status=off");
378                     Electrovalves.setElectrovalvesStatus(2, false);
                }
380                 else {
                    intent.putExtra("LINK",
382                             Electrovalves.URL_ELECTROVALVES_POST_BASE);
```

```
                intent.putExtra("PARAM", "name=EV3&status=on");
                Electrovalves.setElectrovalvesStatus(2, true);
            }
            startService(intent);
        }
    });
  }

/**
   * The function handle the request to toggle the electrovalve number 2
   */
  private void handleToggleEV2() {
      Toast.makeText(this, "Toggle EV2", Toast.LENGTH_SHORT).show();
      mHandler.post(new Runnable() {
          @Override
          public void run() {
              Intent intent = new Intent(MenuActivity.this, HTTPService.class);
              intent.putExtra("METHOD", "POST");
              if(Electrovalves.getElectrovalveStatus(1)) {
                  intent.putExtra("LINK",
                          Electrovalves.URL_ELECTROVALVES_POST_BASE);
                  intent.putExtra("PARAM", "name=EV2&status=off");
                  Electrovalves.setElectrovalvesStatus(1, false);
              }
              else {
                  intent.putExtra("LINK",
                          Electrovalves.URL_ELECTROVALVES_POST_BASE);
                  intent.putExtra("PARAM", "name=EV2&status=on");
                  Electrovalves.setElectrovalvesStatus(1, true);
              }
              startService(intent);
          }
      });
  }

/**
   * The function handle the request to toggle the electrovalve number 1
   */
  private void handleToggleEV1() {
      Toast.makeText(this, "Toggle EV1", Toast.LENGTH_SHORT).show();
      mHandler.post(new Runnable() {
          @Override
          public void run() {
              Intent intent = new Intent(MenuActivity.this, HTTPService.class);
              intent.putExtra("METHOD","POST");
              if(Electrovalves.getElectrovalveStatus(0)) {
                  intent.putExtra("LINK",
                          Electrovalves.URL_ELECTROVALVES_POST_BASE);
                  intent.putExtra("PARAM", "name=EV1&status=off");
```

```
                    Electrovalves.setElectrovalvesStatus(0, false);
                }
                else {
                    intent.putExtra("LINK",
                            Electrovalves.URL_ELECTROVALVES_POST_BASE);
                    intent.putExtra("PARAM", "name=EV1&status=on");
                    Electrovalves.setElectrovalvesStatus(0, true);
                }
                startService(intent);
            }
        });
    }

  /**
     * The function handle the request to show the Drive Electrovalves card
     */
    private void handleViewElectrovalves() {
        Toast.makeText(this, "Electrovalves", Toast.LENGTH_SHORT).show();
        mHandler.post(new Runnable() {
            @Override
            public void run() {
                Log.i(LIVE_CARD_TAG, "State = Electrovalves");
                AppManager.getInstance().setState(ELECTROVALVES);
            }
        });
    }

    private static void setOptionsMenuState(Menu menu, int menuItemId,
                                            boolean enabled){
        MenuItem menuItem = menu.findItem(menuItemId);
        menuItem.setVisible(enabled);
        menuItem.setEnabled(enabled);
    }
}
```

Listing A.25: AppManager.java

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

/**
 * Created by alik on 12/2/2014.
 */
public class AppManager {

    private int state;

    private static  AppManager instance = new AppManager();

    public static AppManager getInstance(){
```

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

Listing A.26: ElectrovalvesView.java

```
package com.google.android.glass.sample.stopwatch;
2
import android.content.Context;
4 import android.graphics.Color;
import android.os.Handler;
6 import android.util.AttributeSet;
import android.view.LayoutInflater;
8 import android.widget.RelativeLayout;
import android.widget.TextView;
10
/**
12  * Subclass of RelativeLayout that is in charge to render the UI of live
 * card when the user wants to drive the valves.
14  */
public class ElectrovalvesView extends RelativeLayout {
16
    private Listener mListener;
18  private boolean mRunning = false;

20  /**
     * Array of TextView for each electrovalve
22   */
    private final TextView[] electrovalveText = new TextView[8];
24
    /**
26   * Interface to listen for changes on the view layout.
     */
28  public interface Listener {
        /** Notified of a change in the view. */
30      public void onChange();
    }
32
    /** Time delimiter specifying when the second component is fully shown. */
34  private static final long DELAY_MILLIS = 40;
```

```
36      private final Handler mHandler = new Handler();
        private final Runnable mUpdateViewRunnable = new Runnable() {
38
            @Override
40          public void run() {
                if (mRunning) {
42                  updateView();
                    postDelayed(mUpdateViewRunnable, DELAY_MILLIS);
44              }
            }
46      };

48      public ElectrovalvesView(Context context) {
            this(context, null);
50      }

52      public ElectrovalvesView(Context context, AttributeSet attrs) {
            this(context, attrs, 0);
54      }

56      public ElectrovalvesView(Context context, AttributeSet attrs, int defStyle) {
            super(context, attrs, defStyle);
58          LayoutInflater.from(context).inflate(R.layout.electrovalves_layout, this);

60          electrovalveText[0] = (TextView) findViewById(R.id.ev1);
            electrovalveText[1] = (TextView) findViewById(R.id.ev2);
62          electrovalveText[2] = (TextView) findViewById(R.id.ev3);
            electrovalveText[3] = (TextView) findViewById(R.id.ev4);
64          electrovalveText[4] = (TextView) findViewById(R.id.ev5);
            electrovalveText[5] = (TextView) findViewById(R.id.ev6);
66          electrovalveText[6] = (TextView) findViewById(R.id.ev7);
            electrovalveText[7] = (TextView) findViewById(R.id.ev8);
68      }

70      /**
         * Sets a Listener.
72       */
        public void setListener(Listener listener) {
74          mListener = listener;
        }
76
        /**
78       * Returns the set Listener.
         */
80      public Listener getListener() {
            return mListener;
82      }
```

```
84      @Override
        public boolean postDelayed(Runnable action, long delayMillis) {
86          return mHandler.postDelayed(action, delayMillis);
        }
88
        @Override
90      public boolean removeCallbacks(Runnable action) {
            mHandler.removeCallbacks(action);
92          return true;
        }
94
        /**
96       * Starts the rendering of electrovalves status.
         */
98      public void start() {
            if (!mRunning) {
100             postDelayed(mUpdateViewRunnable, DELAY_MILLIS);
            }
102         mRunning = true;
        }
104
        /**
106      * Stops the rendering of electrovalves status.
         */
108     public void stop() {
            if (mRunning) {
110             removeCallbacks(mUpdateViewRunnable);
                // mStarted = false;
112         }
            mRunning = false;
114     }

116     /**
         * Updates the view to reflect the current state of animation, visible for testing.
118      */
        void updateView() {
120         if(Electrovalves.isReady()){
               // electrovalves_status = mElectrovalvesStatus.getElectrovalvesStatus();
122             for(int i = 0; i<Electrovalves.ELECTROVALVES_NUMBER; i++){
                    if(Electrovalves.getElectrovalveStatus(i)){
124                     electrovalveText[i].setText("EV"+Integer.toString(i+1)+": ON");
                        electrovalveText[i].setTextColor(Color.GREEN);
126                 }
                    else{
128                     electrovalveText[i].setText("EV"+Integer.toString(i+1)+": OFF");
                        electrovalveText[i].setTextColor(Color.RED);
130                 }
                }
132         }
```

```
        if (mListener != null) {
            mListener.onChange();
        }
    }
}
```

Listing A.27: Electrovalves.java

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

import android.util.Log;

import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.lang.Object;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.concurrent.Semaphore;

/**
 * Static class where the electrovalves status are memorized
 */
public class Electrovalves {


    private static boolean mReady;

    /** Number of Electrovalves */
    public static final int ELECTROVALVES_NUMBER = 8;

    /** URL where the electrovalves information is stored */
    public static final String URL_ELECTROVALVES_BASE =
                                                    "http://CENSURED/show/";
    /** URL where the electrovalves information has to be post */
    public static final String URL_ELECTROVALVES_POST_BASE =
```

```
                                        "http://CENSURED/add/electrovalve";
43      /** URL to set the value of the electrovalves */
        public static final String URL_SET_ELECTROVALVES_BASE =
45                                                  "http://CENSURED/add/EV" ;

47      /** Array where the electrovalves status is stored */
        public static boolean[] electrovalves_status =
49                                          new boolean[ELECTROVALVES_NUMBER];

51      public static boolean getElectrovalveStatus(int index)
        {
53          return electrovalves_status[index];
        }
55
        /** Setter for the Electrovalves Status
57       *
         * @param index , position in the electrovalves_status array
59       * @param value , boolean, true if the electrovalves is on, false otherwise
         */
61      public static void setElectrovalvesStatus(int index, boolean value){
            electrovalves_status[index] = value;
63          Log.i("DRAWER", "EV" + Integer.toString(index + 1) + " has been set");
            if(index == ELECTROVALVES_NUMBER-1) {
65              setReady(true);
                Log.i("ELECTROVALVES", "status ready");
67          }
        }
69
        public static boolean isReady(){
71          return mReady;
        }
73
        public static void setReady(boolean value){
75          mReady = value;
        }
77  }
```

Listing A.28: HTTPService.java

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

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

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

```
11 import java.io.OutputStreamWriter;
   import java.net.HttpURLConnection;
13 import java.net.URL;

15 public class HTTPService extends Service {

17     String httpMethod;
       String httpLink;
19     HttpURLConnection mUrlConnection;
       String mResult;
21     String urlParameters;

23     public HTTPService() {
       }
25
       @Override
27     public IBinder onBind(Intent intent) {
           // TODO: Return the communication channel to the service.
29         throw new UnsupportedOperationException("Not yet implemented");
       }
31
       @Override
33     public int onStartCommand(Intent intent, int flags, int startId) {
           //from the intent I take which http method is it
35         httpMethod = intent.getStringExtra("METHOD");
           Log.i("HTTP METHOD", httpMethod);
37         httpLink = intent.getStringExtra("LINK");
           Log.i("HTTP LINK", httpLink);
39         urlParameters = intent.getStringExtra("PARAM");
           Log.i("HTTP PARAM", urlParameters);
41         new HttpRequest().execute();

43         return START_NOT_STICKY;
       }
45
       // Async task class to make HTTP GET and POST
47     private class HttpRequest extends AsyncTask<Void, Void, Void> {
           @Override
49         protected Void doInBackground(Void...arg0){
               try{
51                 if (httpMethod.equalsIgnoreCase("GET")){
                       URL url = new URL(httpLink);
53                     mUrlConnection = (HttpURLConnection) url.openConnection();
                       InputStream in = new BufferedInputStream(
55                                     mUrlConnection.getInputStream());
                       int ch;
57                     StringBuffer b = new StringBuffer();
                       while ((ch = in.read()) != -1){
59                         b.append((char) ch);
```

```
                    }
                    mResult = new String(b);
                }

                if (httpMethod.equalsIgnoreCase("POST")) {
                    URL url = new URL(httpLink);
                    mUrlConnection = (HttpURLConnection) url.openConnection();
                    mUrlConnection.setRequestMethod("POST");

                    OutputStreamWriter writer = new OutputStreamWriter(
                                mUrlConnection.getOutputStream());
                    writer.write(urlParameters);
                    writer.flush();

                    InputStream in = new BufferedInputStream(
                                 mUrlConnection.getInputStream());
                    int ch;
                    StringBuffer b = new StringBuffer();
                    while ((ch = in.read()) != -1){
                        b.append((char) ch);
                    }
                    mResult = new String(b);
                    in.close();
                    writer.close();
                }
            } catch (Exception e){}
            return null;
        }

    }
}
```

Listing A.29: DataPoint.java

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

/**
 * Created by Buso on 28/10/2014.
 */
class DataPoint {
    private final double mTimestamp;
    private final double mValue;
    DataPoint(double timestamp, double value) {
        mTimestamp = timestamp;
        mValue = value;
    }
    public double getTimestamp() {
        return mTimestamp;
    }
```

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

[13]

LIST OF FIGURES

LIST OF LISTINGS

BIBLIOGRAPHY

[1] Andrea Cavallini Camilla Baj-Rossi Sara Ghoreishizadeh Giovanni De Micheli Sandro Carrara. Design, fabrication, and test of a sensor array for perspective biosensing in chronic pathologies. *IEEE Biomedical Circuits and System Conference*, 2012. URL http://si2.epfl.ch/~demichel/publications/archive/2012/BioCAS_2012_Andrea.pdf.

[2] Texas Instruments. Pcb design guidelines for reduced emi. *Application Note*, November 1999. URL http://www.ti.com/lit/an/szza009/szza009.pdf.

[3] Derek Molloy. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. Wiley, 2014. ISBN 1118935128. URL http://www.exploringbeaglebone.com/.

[4] Avi Baum. An-1852 designing with ph electrodes. *Application Note, Texas Instruments*, July 2014. URL http://www.ti.com/lit/wp/swry009/swry009.pdf.

[5] Dave Evans. The internet of things, how the next evolution of the internet is changing everything. *Cisco Systems*, April 2011. URL http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[6] Charles Severance. *Using Google App Engine*. O'Reilly Media, May 2009. ISBN 0596555806.

[7] Allen Firstenberg Jason Salas. *Designing and Developing for Google Glass: Thinking Differently for a New Platform*. O'Reilly, 2014. ISBN 1491946458.

[8] Jeff Tang. *Beginning Google Glas Development*. Apress, 2014. ISBN 1430267887.

[9] Shalabh Aggarwal. *Flask Framework Cookbook*. Packt Publishing, November 2014. ISBN 9781783983407.

[10] Walt Jung. *Op Amp Aplications Handbook*. Newnes, 2005. ISBN 0750678445. URL http://www.analog.com/library/analogDialogue/archives/39-05/op_amp_applications_handbook.html.

[11] Eric Redmond. *Programming Google Glass, The Mirror API*. The Pragmatic Bookshelf, 2013. ISBN 9781937785796.

[12] Christian Kueck. Power supply layout and emi. *Application Note, Linear Technology*, October 2012. URL http://cds.linear.com/docs/en/application-note/an139f.pdf.

[13] Texas Instruments. An-1852 designing with ph electrodes. *Application Note*, September 2008–Revised April 2013. URL http://www.ti.com/lit/an/snoa529a/snoa529a.pdf.

[14] Texas Instruments. Lmp7721 3-femtoampere input bias current precision amplifier. *Datasheet*, January 2008–Revised December 2014. URL `http://www.ti.com/lit/ds/symlink/lmp7721.pdf`.

[15] Texas Instruments. Lm4140 high precision low noise low dropout voltage reference. *Datasheet*, June 2000–Revised April 2013. URL `http://www.ti.com/lit/ds/symlink/lm4140.pdf`.