Rubén Hermoso Diez

OPTICALLY COMMANDED LAB-ON-A-CHIP CONTROLLER

Bachelor's Thesis

Linköping University
Institute of technology

Supervisor: Dr. Daniel Filippini
Advisor: Carlos Sagües Bláquiz

Linköping, June 2016

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

A mi familia, por confiar en mí y Estar muy cerca a pesar de estar muy lejos.

A mis amigos, por aguantarme día a día y ayudarme a que todo esto fuese mucho más llevadero.

A Daniel, porque además de descubrirme un gran proyecto, me ha descubierto un gran profesor y una persona enorme.

Abstract

This thesis deals with the design, programming and testing of an Android application for controlling a sensor which function is performing an ELISA test. Taking a previous version of the device and the Android application as a starting point. All the necessary steps will be comprehensively detailed, with a strong emphasis on the required concepts.

Contenido

1. Introduction

1.1. Why I Chose This Project and Motivation

When I decided to go on Erasmus and stay abroad during 10 months, I was aware that I had to do the last courses before becoming an engineer and also my bachelor thesis at my destination university.

Finally, I decided coming to Linköping, and here I noticed that there are some differences comparing to my home University in the way that students can finally get a thesis for finishing their studies. Here, it is needed to find a teacher, discuss with him the topic and then accept it or not, there are few options which can be found on the web, but most of the teachers prefer discussing it on their departments.

In my case, I was very interested in microcontrollers, programming or electronic instrumentation, so I started looking for some available projects related to that. The fact was that I could not find anything interesting.

After some weeks, and thanks to Professor Daniel Filippini I found my topic. I was surprised, as it was implementing a controller for a sensor, but the platform where the code was going to be executed was not a microcontroller, it was an Android mobile phone, and I was completely unfamiliar with this field. At first I was a little doubtful about accepting it or not, but finally I decided that it could be a good idea, as the use of the phones nowadays is very extended and I was quite curious about Android programming. I also thought that maybe it is not a crazy thing using mobile phones for controlling complex devices or sensors if they do not need to be static in a place.

Finally accepted it, as it was a great opportunity for discovering new fields and being aware of the huge possibilities that a mobile phone can contribute for controlling and automate a process.

1.2. Goals of This Project

Once I accepted the project, the next important step was being aware of what I had to do, what my function in that researching group called ODL (Optical Devices Laboratory) was. In short, this thesis deals with the problem of designing and programming an Android application with the aim of controlling some valves, a pump and some extra outputs integrated on a device for achieving a successful ELISA test inside a Lab-On-A-Chip.

The application I had to design and program was an update of a similar application started by Professor German Comina. At the very beginning it was certainly helpful having his starting application for giving me the idea of my first steps, despite the final app is completely different of this starting one in both graphic and logic design.

In my opinion, this project is not a complex one for somebody who has the previous knowledge needed for programming an application in Android, and some experience using JAVA, but for me (novice at this field) it was a demanding exercise. But as I see this, it brought me the opportunity of being able to reinforce the idea that no matter what you are asked to do, making some effort and working constantly it can be achieved, and also to discover a new field where now I feel really comfortable with, that is Android programming.

1.3. Objectives

The goals of the project, both theoretical and practical are the following ones:

- Redesign of the optical coupling stage of the existing platform, including the mechanical vehicle to support sensors and cell phone attachment, for which ODL is equipped with a Form1+ 3D SLA platform.
- Understanding the previous application, redesigning, and adding some new functions for the upcoming changes.
- Design of the feedback stage and controlling software. Measurement carried out on a LOC, created with the fast prototyping unibody-LOC (ULOC15) technique developed at ODL
- Characterization of the measuring systems of the control sequence required for an immunologic LOC configuration.
- Writing the Project Document.

1.4. How This Project Was Developed

	March	April	May	June	July
Study of previous documentation	X				
Redesign of the platform	X	X	X	X	X
Redesigning the application		X	X		
Design of the controlling soft			X	X	
Characterization of measuring syst				X	
Writing the project document				X	X

Table 1 Chronology of the project

The default load of work established for this thesis was 22.5 ECTS, which was readapted to my needs, around 18 ECTS of real work load. Taking into account that an ECTS is equivalent to 25 hours of work, the estimated time for accomplishing my project was 450 hours in total. Distributed approximately as it is shown in *Table 1*.

This thesis started on February 2016 when I was given the topic, but in the real context, the project didn't start until March, when I really started working on all the tasks and being aware of what I had to do. The development of this project finished in the end of June of the same year, when the application was completely finished and only this document was left to finish the bachelor thesis.

Apart from this, also there were several reuinions with my supervisor Professor Daniel Filippini, who was always up for a meeting if I needed it and really helped me to keep working with the project.

2. Background

2.1. ELISA test

The ELISA test (Enzyme-Linked ImmunoSorbent Assay) is a test designed by Swedish and Dutch scientist in 1971 that uses antibodies and color changes to identify a substance. The technique consists in using an antigen immobilized (that uses to be a protein fragment), detected by an antibody linked to an enzyme capable to produce a detectable product like a change of color. If colorants are used, it is possible to measure the antigen directly by using spectrophotometry, as the amount of color indicates the amount of substance present.

Despite it is a routine technique and quite easy, it involves a big amount of variables, as the temperature, volume, time or reactive selection, and if they are not adjusted correctly, next steps of the test can be affected.

In medicine this test is used to identify germs which can be found in blood, urine, sputum… Also can be used to determine if a patient has any autoimmune disease or infection. This technique was generalized and used with simple and cheap devices for diagnosis. Despite all the advantages mentioned, it also has some limitations:

- A positive result that confirms the presence of antibodies does not mean that the patient has a disease. A person who has been ill and now is recovered can still be producing antibodies (false positives).
- On the other hand, a person who produces less antibodies than normal can have the opposite problem, as these antibodies can be unnoticed if they are too few, causing a false negative.
- If there is not good affinity between the antigen and the antibody, this means that there are weak links which can create false negatives in the test.

Several devices have been used for this tests, at the beginning until the microplates used nowadays which have 96 holes made from special plastic for improving the adsorption, and with the bottom of the hole made of transparent plastic for making the measurements easier.

2.2. Biosensing with cell phones

Nowadays that mobile phones are comparable to a mobile computer with some physical sensors embedded in it, such as magnetometers, accelerometers, gyroscopes, proximity sensors… Which can be used for example to know how our mobile phone is oriented, or for turning off/on the screen of the phone while we are talking through it. All these functionalities, which are quite effective, are useful as improve functions of the phone, but for the moment, mobilephones do not support chemical sensing or biosensing. Measurements cannot be done directly, so some extra devices are used complementing these physical sensors named before for making biosensing possible by using a mobile phone.

Biosensing and diagnostics tipically involve the detection of analytes in solution and sample conditioning, these two things are impossible to be made with the phone, but they can be integrated within a Lab-on-a-Chip (LOC).

The use of disposable fluidics with detection chemistry simplifies aspects such as calibration and can exploit chemical iteractions. Also biosensing with phones exploits the physical-sensing capabilities and the processing power of the mobile phones to provide the support required for advanced biosensing. But this is not the first time that consumer electronic devices (CEDs) have been used for complementing biosensing, such as flatbed scanners, CD units, or web camera-screen combinations.

Nowadays that mobile phones have their own OS, they are the most common devices used for biosensing with minimal or no dedicated interfacing electronics, two strategies can be identified:

- Use of auxiliary reusable devices (ARDs) specifically designed for a phone brand and used for link the biosensing assay to the cell phone.
- Auxiliary disposable devices (ADDs) have generic designs that are compatible with diverse phone brands and models. For our project we used this kind of device, since the idea was being able to make ELISA tests with no dependence of the mobile phone we were using. In our case a Samsung Galaxy Note II.

Another helpful physical sensor that mobile phone owns is their cameras, ADD and ADR biosensing are dominated by optical detection, this supported by the continuous development of the phone cameras and the optical biosensing helped with LOC solutions.

Biosensing techniques often exploit optical coupling to cell phone cameras, which are present in approximately all the phone models nowadays, as well as image and video acquisition become more and more popular for recording phenomena happened during the tests as chemical responses, for example detecting a change of color, displacement signals, contrast… In addition, this camera can help for having control of the test closing the control loop optically doing some actions when these responses are detected, or even fixing concrete volume, creating a region of interest.

2.3. LOC Devices for cell phone biosensing

It can be said that the same instruments interfaced to computers can be implemented with cell phones, which gives certain freedom to the instrumentation design. But the main objective on this project is preserving the cell phone's ubiquity by adapting the sensing. These devices should be low-cost and adaptable to any phone brand/model. Now it is going to be analyzed the lab-on-a-chip (LOC) biosensing that can be adapted for this aim.

One Lab-on-a-chip is a device which integrates one or several functionalities which can be done in a laboratory in only one chip. One Lab-on-a-chip allows to work with volumes extremely reduced with a huge accuracy, even less than picoliters, this makes Lab-on-a-chip a very useful tool for working with microfluidics.

For this project a Lab-on-a-chip printed with a 3D printer is used. The reason is that, during the development of a project, it is needed to make some trials with the LOC, for adjusting the design to the necessities, and sometimes the design can be improved, and it can be only noticed by testing (for example using new materials or different pipelining), so a "cheap" and very fast way of obtaining this prototypes was 3D printing.

In addition, another thing that must be taken into account that the size of the LOC is very small, so for the 3D printer there are some troubles when the top cover needed to be printed, as the shape of the design in the actual printing was not corresponding to the one showed on CAD. So Professor D.Filippini decided to leave the device open on the top, leaving the pipes made without the top part, and covering it with transparent scotch tape for a correct operation of the device.

3. Study of the Previous Application

Once I was informed about the aim of the project, and I did know the basic theory concepts and approaches, the next step was understanding the actual state of the project and, how the previous Android app worked.

This project involved software and hardware development, despite my main goal was developing the control application, I also had to understand the electronics and how the device worked for reaching my objective. Some hardware and design modifications were performed during the development of the project for optimizing the device and adapting it to the new possibilities that a more advanced application offered.

The first objective was to know how the project was raised: for performing a conventional ELISA test, it is needed that one person does the optimum steps and revises the results obtained for reaching a diagnosis. One of the biggest problems is the amount of time and resources needed, since normally the tests are performed in boards with lots of small holes where everything is carried out. The objective of this project was eliminating all these problems integrating all the tools needed for performing this medical test into a self-dependent device.

3.1. Hardware

For making this possible, professor D. Filippini and G. Comina decided to use a 3D-printed Lab-on-a-chip, inside it all the reactions and liquid substance mixtures will happen and some results will be obtained and analyzed by the experts later on. This Lab on a chip would be located onto an electronic platform composed by various valves and pump, once the test is finished, the pipes would be empty and cleaned with water for restarting the process automatically. More concretely the device was composed by:

- A switching regulator responsible of supplying power to all the electronic devices, the valves and the micropump that will be described in the following steps.
- For making the device self-dependent in an energetic way, a LiPo Charger was included, using a 2100mA battery for having a reasonable autonomy. In later steps we decided to quit it due to some bad connections.

- The LOC where all the chemistry is carried out, this LOC allows to handle low volume fluids and was the key element for carrying out the ELISA test.
- A very simple hydraulic circuit composed by two simple effect valves which output is followed by two double effect valves that allow four different outputs for the fluidic substances. For pumping the water through the pipes a Micropump mp6-pp was used, this micropump is intended for pumping liquids with varying flow rates controlled by electronics. For driving this micropump it was used a L9110 2-Channel Motor Driver as it is shown in the next figure:

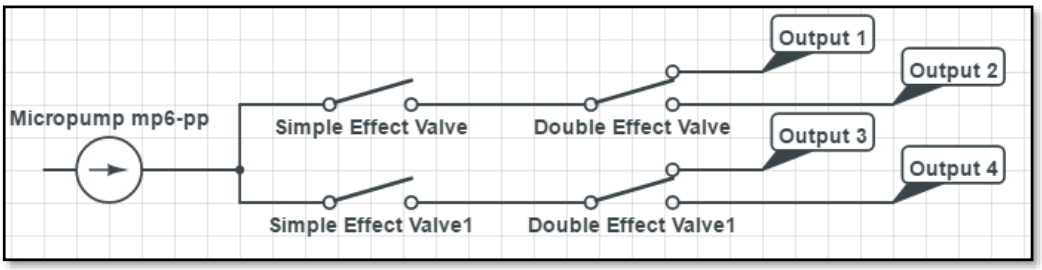

Figure 1 - Simplified Hidraulic Circuit

Next step was choosing a way for controlling all the items described and making them work together for performing the tests, conventionally computers are the most common tool used for doing this task due to the processing capacity and the big range of applications and programs that can be used for the control aim (Matlab…).

For our concrete case, one of the main goals of the project was doing a portable device, that could be used almost anywhere without the need of being physically connected with a cable to any other apparatus like computers. For achieving this, the input signals and instructions given to the device would be optical signals generated by a mobile phone: some light sequencies were shown in the mobile phone screen and capted by 8 photoresistors to control the valves and motors. Depending if the screen showed a black square, which meant ACTIVATE, or a white square used to DEACTIVATE the different independient elements (the pump, the valves…), also some extra features where implemented for a future possible use: aux and LED which use is not given yet in the project.

Four different sequences were used for selecting one of the four possible outputs in the valve circuit, as it is shown in the next figure, where the photoresistors illuminated with white light would be the ones represented by a circle filled by blue. P and Pc are related with the pump, for activating it is necessary to have both of them with no illumination, otherwise the pump will not work at all:

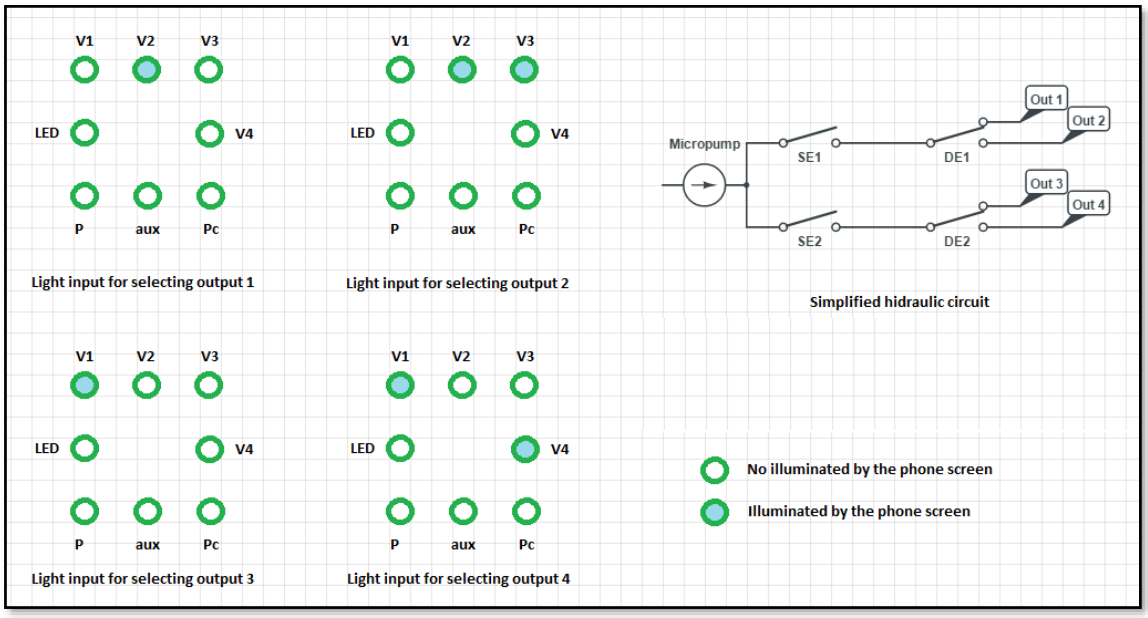

Figure 2 – Illumination scheme for enabling pumping fluid through different valves

3.2. Software

After understanding how all the hardware part worked it was time to deal with the software. Originally the control application designed for driving all the electronics and performing the tests was designed using Adobe Flash CC, this program is typically used for developing games and applications focused on the graphical part. The language used for programming was Adobe Action Script.

Previously, a timed control was used, this means that the sequences shown on the phone screen to be detected by the photoresistors changed each a determined quantity of seconds.

The basic operation of the application was: when the user pressed the start button, the app was initialized and the 8 squares were shown on the screen that will be in contact with the photoresistors. Next step was loading the sequences that will be used for the test, which were contained on a LibreOffice file uploaded to the Linköping's University server. After this the different columns were split for obtaining the data of each sequence, then the corresponding frames were shown during a determined amount of time which was determined in the Server's LibreOffice. When the process was finished, the application was ready to start again by pressing the start button. This app could be stopped immediately if the stop button was pressed.

3.3. Defining work lines

First of all, the program used for developing the application was Adobe Flash CC, this was a big disadvantage as it needs a license that Daniel had to purchase if this program was needed. The whole previous app had been programmed using the trial version of the Adobe Flash CC. One of the first things I started checking was the viability of developing the app using some other program.

I found out two possible options: Oracle and Android Studio. With both of them it was possible programming Android applications, and the programming language was JAVA. At the end, since the installation and the graphic interface of Android Studio looked better for me, I decided to start using this program for redesigning the application.

Dr. D.Filippini also commented some modifications for including in the new application:

- Finding a way to not depend on the internet (as the sequences had to be stored on the server) for performing the tests in the timed application, for using it while the optically controlled app was being developed.
- The main focus of this thesis, which deals with the problem of controlling: pumping fluid during a determined period of time for fixing the volume used for the tests is quite uncertain, as maybe the response of the micropump is not all time the same. The LOC works with very small volumes of fluid and it was quite hard to be sure that for all the test the volume used was exactly the same and was adequate if the control was timed.

4. Things Learnt

4.1. Files and exceptions

If there is a need for storing Data into memory, two different ways of storage can be chosen. RAM memory, which is fast, volatile and it has a quite limited size of storage. But sometimes, this information is too big for being stored at RAM memory, or maybe we need to keep it during a long time, even if the device is turned off and restarted. This data is stored in files, a collection of information contained into a physical support which can be volatile or permanent, the data type must be the same during all the document (integer, string…). Depending on the information contained we can classify the files into two groups. Binary files which are the ones that hold binary digits inside. And text files store alphanumeric characters with a standard format (ASCII, Unicode…) and can be modified by text editors. The operations for handling a file are:

1. Creation (if the file did not exist before)
2. Opening the file
3. Editing the file (reading/writing, erasing…)
4. Closing the file

When it is needed to communicate our program with another data source or container, streams are used, which are threads where data flows freely, and can be input streams or output streams depending on our interests. They are grouped at *java.io* class. The steps for using streams are mainly the same: creating, reading of writing and closing it. The *Reader* abstract class is used for reading characters, depending on the data source, different subclasses are used, which are shown in the next table.

Source	Class
Buffer	BufferedReader
File	FileReader
Char array	CharArrayReader
Stream that can receive characters once is read	PushbackReader
Buffer for counting lines	LineNumberReader
String	StringReader
Byte string	InputStreamReader

Table 2 - Reader Classes

There are also some basic methods tipically used in the Reader class:

- Int read () : reads the next available character in the stream. If there's no more characters available, it returns -1.
- Int read (char [ ] v) : tries to read as many characters as v array's size. This method returns the number of characters read. If there is no more characters available to read, it returns -1.
- Void close() : closes the stream

Classes used to write characters are derived from the abstract class Writer. Depending on the data destination, different classes can be used, as it is shown in the following table:

Destination	Class
Buffer	BufferedWriter
File	FileWriter
Character Array	CharArrayWriter
Formatted Text	PrintWriter
String	StringWriter
Byte string	OutputStreamWriter

Basic methods used in the Writer class are:

- Void write (int c) : writes the “character” contained in c
- Void write (string s) : writes the String s
- Void write (String s, in pos, int len): writes len characters contained in the chain s, from the position pos.
- Void write (char [ ] b): writes in the stream the char array b
- Void write (char[ ] b, int pos, int len) : writes len characters contained in b, starting in the position pos.
- Void close() : closes the stream.

In case there is any problem with the input or the output, all the methods launch an IOException.

For accessing to a file, there are some classes, FileReader is similar to reader, but it is used for reading text files. There are two useful methods, FileReader(String name), which is a constructor whose argument is the name of the file, which is received via a string, and the FileReader(File name), which is a constructor with a File argument, which is the file that is going to be read. Also some exceptions are launched just in case that the file does not exist. In case it is needed to write into a file, FileWriter (String name) can be used, using the name of the file as an argument, or simply indicating the file it is going to be read using FileWriter(File name).

Appart from that, files can be edited, for this, PrintWriter class can be used. The most common methods for writing into a file are println(String s) and print(Strings).

4.2. Digital image concepts

Nowadays, information is suffering a digitalization process where images play an important role. Photos, graphic design, cinema, TV… thousand images are produced and stored into a physical support, some others are sent, projected or printed.

When it is needed to deal with images, and process them after, it is necessary to take some decisions, for example, the kind of compression, the size of the file, the quality of the image once it is stored…

First step for example is deciding to produce a vectorial image or a bitmap. Since this two types of images are created an edited with different programs, and they have completely different uses.

4.2.1. Vectorial images

Vectorial images are composed by simple geometrical identities, segments and polygons (in fact, a curve can be expressed as a succession of segments). Each one of this entities is defined mathematically by a group of parameters (initial and final coordinates, thickness and color contour, fill color… With this technique, quite complex images can be designed.

One of the most useful advantages of this kind of images is that as it is composed by simple geometric entities, this images can be resized for enlarging or reducing them without any loss of quality. This way, plain colours and clean contours are maintained no matter the size they are shown.

4.2.2. Bitmaps

Bitmap images are built using a huge amount of little squares called pixels, each square is filled by a uniform color, but the sensation perceived by the eyes is the result of integrating the color variations and the luminosity between neighbor pixels.

This bitmap images are very useful for representing illuminations or big tonal variations into a scene. In fact, this is the type of image used at photography and cinema. Obviously the quality of the image varies depending on the quantity of pixels used for representing it.

4.2.3. File size

The size of a file is a value, measured in bytes or bits that describes the amount of memory used for storing the image information into a physical support (USB, SD card…). This size depends on several factors, especially the resolution (R), the image dimensions (Length x Width) and the color depth (D). With this information, the size of a file can be estimated by the following formula:

$$File\ Size = R^2 \cdot L \cdot W \cdot D$$

For the correct interpretation of this formula, it is needed to express the length units using inches, as it is shown in the next formula:

$$File\ Size = (ppi)^2 \cdot inch \cdot inch \cdot bits = bits$$

It is important to know that 1 byte is equivalent to 8 bits and 1 Kilobyte is equal to 1024 bytes. As it is shown, the resolution influences the most in the final file size as it is squared, one of the most important steps is choosing an optimal resolution, adequate to the needs of the project for reducing as much as possible the size of the file it is created, since the storage space is not unlimited and not always a high quality image is needed.

The resolution is the capacity of reproducing faithfully the details contained into an image. For a bitmap, this resolution is referred to the quantity of pixels that form that image. It is measured in pixels per inch (ppi). The more resolution, the more image quality.

On the other hand, the color depth is related to the number of bit which are used for describing the colors contained in each pixel of the image. With a high color depth, there are more color possibilities and the representation can be more precise and have more shades. In the following table there is a calculation where it is shown the amount of colors available for each depth:

Depth	Colors
1 bit	2
4 bit	16
8 bit	256
16 bit	65536
32 bit	4294967296

In a bitmap, the quality is composed by various layers, one for each basic color (red, blue and green) and another one for the luminosity (complete darkness to complete light).

Above 16 bits of depth, the color description is divided in layers. If the depth is 16 bits for example, 4 bits (128 levels) for each layer are used. If the depth is 32 bits, each layer uses 8 bits (256 levels) to adjust the colors.

4.2.4. Color mode: RGB

The eye perceives colors depending on the wavelength that receives. White light contains all the color spectrum, while the absence of light is perceived by our eyes as the black color. Most of the edition programs use several color modes for defining and classifying all the possible colors. Most of the programs use HSB (hue, saturation and brightness), RGB (red, green and blue) or CMYK (cyan, magenta, yellow and black).

RGB mode is used in all processes where color is obtained by additive mixture of lights: television, graphic screens, artificial illumination… In all this devices, the whole color range is obtained by mixing three primary colors: red, green and blue.

In this case, any color is obtained mixing two or more lights, the mixture of variable proportions of colors produce the complete color range. The mixture of the three basic colors produces white color, while the absence of colors produces black color. The image edition applications, express the quantity of each primary color used for defining the pixel by using a number that can adopt a value from 0 (absence of color) to 255 (maximum). For example, one RGB color can be defined with the digits (156, 69, 242).

4.3. Android Studio Basics

Android is an OS based on Linux Kernel designed mainly for mobile phone devices with a touch screen. Originally, Android was developed by Android Inc, a firm bought by Google in 2005. In 2007 it was presented to the public and since that date it kept growing until today.

As a starting point, Android was originally developed for smartphones, buy nowadays with this crescent development of application and technologies, this operative system can be found even in watches, vehicles, televisions…

The main characteristics of Android that are interesting for this project are, among others: the adaptation, since Android applications are adjusted to different screens without any problem. There is also support for additional hardware, since this OS supports cameras, tactile screens, GPS, accelerometers… And related to this, Android also allows working with some different multimedia formats for video and audio without any problem. With all this characteristics, taking into account that supports JAVA and the development environment of this platform is quite easy and useful (there are 2 official ones: ADT and Android Studio) and provides an emulator for testing the apps developed.

4.3.1. Android graphic interfaces

One of the most important features in Android Studio is the graphic interface into its apps. For analyzing this it is important to clarify some concepts that is important to know before:

The Views are the basis in the graphic development in Android, all the graphic elements are subclasses from View. Other complex elements, as view aggrupation or layouts are inherited from this class. This way, in Android, a button or a text field are views. One of the most common used views are:

- TextView: it is a simple text label which is used for showing a text to the user. Its most important attribute is Android:text which value indicates the text shown in the screen. Other useful parameters are android:textColor and android:textSize.
  Inside the code, the most used methods are setText and appendText that allows modifying the text shown or adding additional text during execution. In both cases the parameter received is a string.
- EditText: it is a subclass prepared for text editing. When the view interface is pressed, a keyboard is shown, allowing entering data. The code is managed the same way as with TextView by using for example getText or setText.
- Button: which represents a normal button with a text associated to it. For changing the button text, android:text must be used.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
      <TextView
              android:id="@+id/texto"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Texto"
```

```
      />
      <Button
              android:id="@+id/boton"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Púlsame"
      />
</LinearLayout>
```

The easiest way of interacting with a button is pushing it for unleashing an event. Also another function must be implemented for capturing the click event (OnClick) as it is shown in the following code:

```
public class MiActividad extends Activity {
      protected void onCreate(Bundle icicle) {
              super.onCreate(icicle);

              setContentView(R.layout.miLayout);

              TextView texto = (TextView)findViewById(R.id.texto);

              Button boton = (Button)findViewById(R.id.boton);

              boton.setOnClickListener(new OnClickListener() {
                      public void onClick(View v) {
                              texto.setText("Botón pulsado");
                      }
              });
      }
}
```

Another option is defining the handler for the click on the XML file, by using the android:onCLick attribute. The value of that attribute will be the name of the method that will handle the event, once the button is pressed, the method will be launched.

- ImageView: this subclass displays an image, it can be loaded from various sources (such as resources or content providers) and it can be used in any layout manager, also provides different options such as scaling or tinting. As when buttons are used, the first step is assigning an ID to the imageView and then the next step is modifying the different editable properties as it is shown in the following code:

```
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/android"
     />
```

For associating this XML code with the android Activity it must be used *image = (ImageView) findViewById(R.id.imageView1)* and for displaying a concrete image placed in the drawable file it is used the function: *image.setImageResource(R.drawable.myImage)*.

Layouts are the means used to organize views in the device screen, there are different types of layout that allow us dispose elements on the screen in different modes. Using layouts allows separating the logic and the design parts of the application, with flexibility enough to modify the graphic interface without need of modifying code. Some of the available layouts are:

- FrameLayout: this is the simplest one. It adds the views on the top left corner. If various views are added, they are superposed.
- LinearLayout: aligns different views on a horizontal or vertical line, a vertical linear layout consists on a column of views and a horizontal linear layout is a row of views. These layouts allow establishing a weigh to each element, which controls the relative size of each element inside the view. A basic example of use can be found in Annex B.
- RelativeLayout: it is the most flexible layout, that allows defining the position of each one of its views relative to the position of the rest of the view or the corners. An example can be found at Annex C.

Defining the graphic interface by using resources instead of code, also allows to specify different layouts depending on the hardware configurations or even the interface can be modified while execution when an event is produced, for example when the screen orientation changes.

Alo t is very common that one application has more than one screen for increasing its functionalities. For implementing this in Android it is necessary to include two more files: one XML file with the graphic interface of the new screen and a JAVA one with the logic part.

After creating a new activity for doing this, there will be two new files corresponding to the ones that were created when the project was created: new*activity_main.xml* and *MainNewActivity.java*.

Instde the very first Main activity it is needed to create an Intent class object where the parameters used are the references from the object of this class, and the reference of the new activity (NewActivity.class). After it is needed to call the method *startActivity()* for displaying on the screen the new activity created. This can be seen in the following example code:

```
public void newActivity(View view) {
        Intent i = new Intent(this, newActivity.class );
        startActivity(i);
    }
```

4.4. Managing Android Camera

In this section it will be discussed how to manage the android camera, and how to display on the screen a camera preview where the images capted by the camera can be seen in real time.

First of all, it should be clear that it for some uses, where integrating photos or videos is not an important part on the application developed, the default camera application can be used. For our purpose this is not possible due to some limitations, as we need to access to data generated by the preview of the camera in next steps and this cannot be done using the simple camera intent.

When the camera user interface needs to be customized to the look of the existing application, and providing special features it is easier to build an own camera application for obtaining a more compelling experience. During this section the necessary steps for creating a custom camera interface for an specific application will be explained:

- Detect and Access the Camera: in this step it is checked if any existing camera is detected by our mobile phone. Also it is possible that the device used has more than one camera (if it has front facing camera). On the last Android API it is possible to check the concrete number of cameras available on a device using *Camera.getNumberOfCameras()*.

```
/** Check if this device has a camera */
private boolean checkCameraHardware(Context context) {
    if
(context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){
        // this device has a camera
        return true;
    } else {
        // no camera on this device
        return false;
    }
}
```

After this, access to the concrete camera must be requested, for making this is needed to call *Camera.open(int)* where the number given to the function is corresponded to the camera which access is demanded (0 for backfacing camera, 1 for frontfacing camera…). Once it is achieved, some camera parameters can be shown and changed by using *Camera.getParameters()* there it is possible to select the focus mode, rotate the image, enable or disable the flash…

```
/** A safe way to get an access to the Camera. */
public static Camera getCameraInstance(){
    Camera c = null;
    try {
        c = Camera.open(); // attempt to get a Camera instance
    }
    catch (Exception e){
        // Camera is not available (in use or does not exist)
    }
    return c; // returns null if camera is unavailable
}
```

- Create a preview class: for users to effectively take pictures or video, it is needed that they can see what is capted by the camera. The data can be displayed live since the camera preview class is a *Surfaceview*, and the users can frame and capture pictures or videos. In addition, *surfaceChanged* can be used to modify the specific size of the preview. For all these things it is needed to create a specific class similar to the one included in the Annex A.

4.5. Processing Images with Android Studio

When it is needed to process an image with a computer, some specific programs are used. For working with android Apps the image processing can be done directly using Android Studio libraries but in case this is not enough, it is also possible using OpenCV which is free artificial vision library originally developed by Intel, which offers huge possibilities for processing video and images captured with a camera.

For most programs, images are stored and represented as bidimensional vectors (matrices) where each element is corresponded with a single pixel. Working with images in some programs like Android Studio or Matlab is equivalent to work with the data type matrix. Depending on the type of image used the data contained inside the matrix can be different:

- Binary images only contain two possible pixels: black and white. The representation of these images is made by a matrix with only two different elements, 0 and 1.
- Grey scale images: where only one matrix is used for representing the data, but for this case, the elements can acquire up to 256 different values corresponding to the different grey levels.
- RGB images: in this last case, the final image is composed by three bidimensional matrices corresponded to different layers R (red), G (grey) and B (blue). Into each layer the elements can take numeric values up to 256 for representing the colors as explained in previous sections.

For this project, we will focus on RGB images. A useful thing to know is the possibility of going across the different elements of the matrix for checking the colors that are contained into pixel. For doing this, the different layouts (i, j, k) will be checked, and for each of the layouts (R, G, B) a value corresponded to a concrete intensity will be obtained. This is implemented in the bitmap library with the function: *getPixel (x, y)* where the RGB color from a specified location is returned as an integer and can be stored into a variable.

While going through the different elements contained into a matrix, it is also possible to discard some layers if needed, in some medical cases where liquid substances are used, especially ours, it is very useful since the fluid can be pigmented with a distinctive color which can be completely different from the rest of the colors contained in the image, this way if some layers are discarded it is easier to find a layout where this pigment used in the fluid will be highlighted comparing to the rest of the image. For doing this in Android Studio, it is needed to call the Color class, where the different layouts can be selected calling *Color.red(pixel), Color.green(pixel)* and *Color.blue(pixel).*

It is also useful to know some functions used for creating a bitmap specifying different parameters using *createBitmap()* and selecting its height and width using *setHeight(int)* and *setWidth(int).* For modifying it changing for example the color of some pixels *setPixel(int x, int y, int color)* must be called, where the specified color is written into the bitmap at the x, y coordinate. More information can be found in the Android Studio Developer API, looking for the specific class Bitmap.

5. Applications Designed

As it was said during this document I had an application implemented using Adobe Flash CC as a starting point. I started modifying this application for showing the camera preview on it and I obtained some good results as it can be seen in the next figure. The code associated to this can be found at the Annex E.

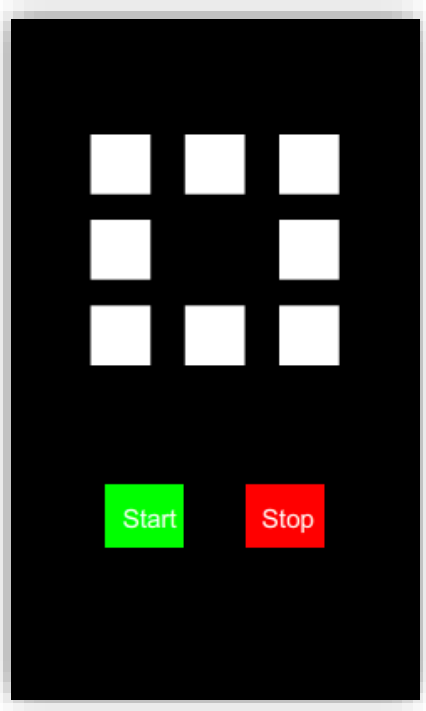

Figure 3 - Given application

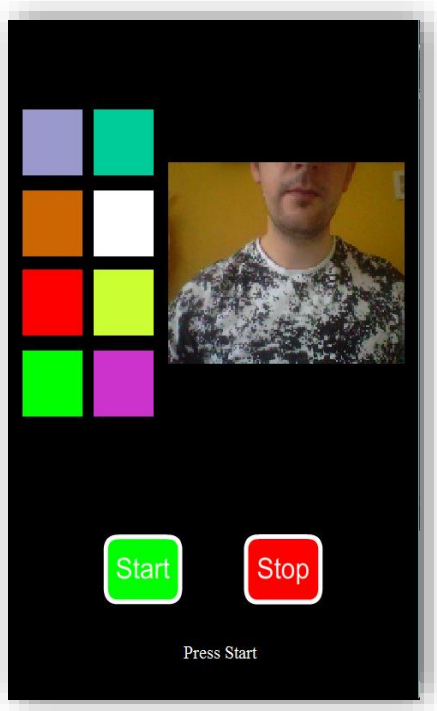

Figure 4 - Modified application using Adobe Flash CC

There were some graphic and logic modifications comparing to the original app and it was possible to select the camera displayed between the front facing and back facing camera in case that there were more than one available.

Also I moved the squares that in the original app occupied the entire screen, displacing them to the left for fitting the preview, this last thing involved a hardware modification since the original device was designed for being placed above the phone screen and the photoresistors were originally fixed to a position that had been changed in the last app version, for solving this, we decided to wire the photoresistors using long wires, and covering them with an independent case for making the light detection more flexible as we could choose where to put the device, that now did not need to be above the phone screen as it can be seen in the next figure.

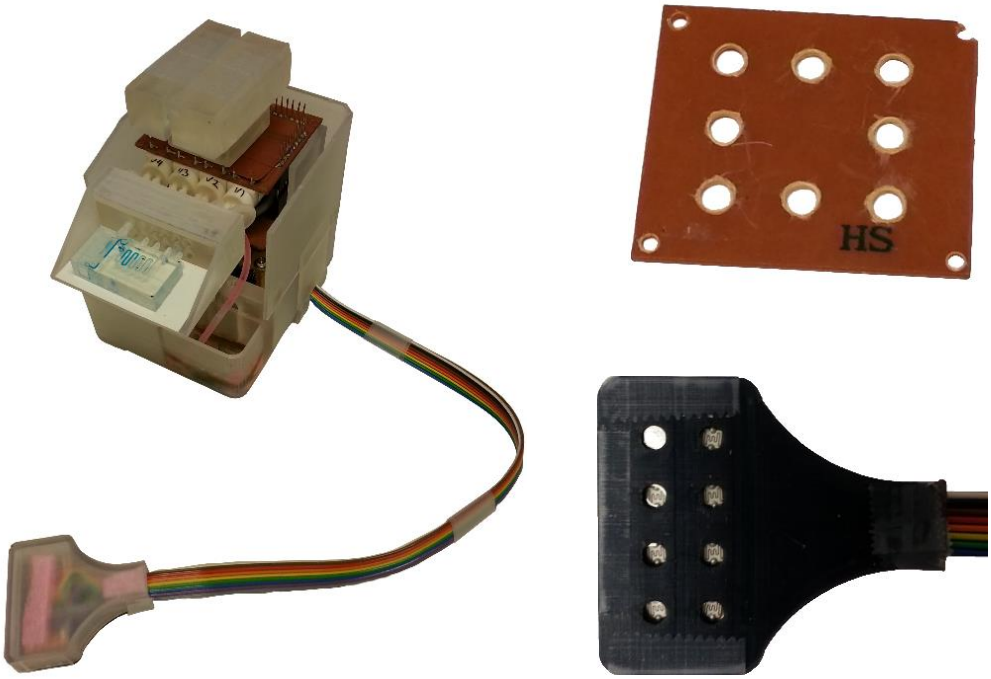

Figure 5 - Device and modifications made for adjusting photoresistors to the new app. Previous method on the top right

While developing the android application using I was really surprised since Adobe Flash CC looked quite easy and also I had not so many problems for making the camera preview appear on the screen.

Problems started when I tried to go on with the next steps of the project, where I needed to access to the mobile phone camera preview information for a further processing of the images. I also had never used Adobe Action Script for programming so I started thinking on the possibility of using another Android development software more adapted to my needs.

After informing myself about different options, I finally chose Android Studio, as it was designed specifically for developing Android apps and was completely free, offering a lot of possibilities and a very comfortable graphic interface. Its installation was also easier than other options like Oracle.

5.1. Timed Process

First step was being able to migrate the previous application to Android Studio, it may seem like a simple task, but it wasn't as I had never used Android Studio before, so I started learning basic concepts of Android, once I more or less understood how everything worked, was time to start with the new application.

Daniel also suggested me to store in the phone the sequences corresponded to the light illumination, as before this data was retrieved from the Linkoping University's server. For solving this inconvenient, first of all I decided to save the different sequences on the device storage in order to access to them later and show the corresponding squares without the need of internet.

This timed application consists in 3 different screens, one of them corresponded to a selection menu, where the user can choose between creating or opening a sequence, and the control mode where that sequence is used to control the device with the illumination code.

In the second screen, where new steps can be added, first of all it is needed to introduce the *File Name* (which should be different each time, otherwise the information will be overwritten in the same file all time) followed by the number of steps that will be executed, and a final field where the instructions of each sequence are entered by the user, this last field will determine which squares will be illuminating the photoresistors. For each step, the following information must be included:

1. Valve 1: 1 if square HIDDEN or 0 if square SHOWN.
2. Valve 2: 1 if square HIDDEN or 0 if square SHOWN.
3. Valve 3: 1 if square HIDDEN or 2 if square SHOWN.
4. Valve 4: 1 if square HIDDEN or 2 if square SHOWN.
5. Pump: for enabling the pump, both squares related to it must be HIDDEN, corresponded to a '1' in the code, for disabling the pump a '0' is needed.
6. Time: expressed in seconds, this time must have two digits, since for the tests performed at that moment we did not need to deal with big amounts of time.
7. LED and AUX were two not used variables in this code due to the changes in the application and the platform. Daniel decided to leave them as auxiliary photoresistors for future uses.

Once all the data described before have been introduced, the user must click on the *Save* button for storing a text file with the name chosen, this file contains all the information written. It is important to know that the numbers introduced by the user are saved on a single line, this means that each character is only separated by the rest by using a simple space no matter if some line breaks are introduced by the user (will be erased). For example:

```
Step 1


Sequence 1 0 1 1 1 09 1
```

Note that also the number of steps that will be executed is stored in the first position of the text file, as it can be seen in the following example where it is shown how the information is stored into the text file:

```
1 1 0 1 1 1 09 1
```

At the same time that the first file is created, another one with a predefined name is automatically created/overwritten, this new file is used in the other activity (control screen) where it is read by default. Reading all time the same file avoids the doubt of which filename should be loaded, as it will be all time the same.

For editing a text file already created or just check the content of the file, the user needs to introduce the original name given to the file and click *Load*. It is important to know that the control program will only run the last file stored, this means that if for example “Test 2” is being used, but it is needed “Test 1” for some trials, this “Test 1” must be loaded and saved again for being ready to use.

When *Stop* is clicked the process finishes, this ending is not immediate, it is needed to wait for some time corresponded to the time of the last step which was being executed. Once the time is up, a popup message is displayed informing that the process has finished correctly and it is possible to restart again the test.

Figure 6 - Main menu, Add new file and Test screens

It is also possible to turn on the flash. It was also useful for preventing a bad illumination while using the back facing camera, in that moment we did not know where to place the phone for dealing with the detection, and also did not know about the conditions needed for performing the tests, so it was a good idea to implement that option. Finally, we used the back facing camera, and decided to place the mobile phone above the device, where the LOC was focused by the camera of the phone, giving also a chance for illuminating with the flash.

In this timed application, I also implemented the option of recording a video of the test while it was being performed. In that concrete moment I was not aware of how to access to the information related to the camera preview, so I started recording a video just to try to process it later.

This application was programmed as a compendium of different blocks which were programmed independently, and after that I made them work together for achieving a result, the different blocks used for design this program were:

- *SaveTextFileOnMemory*: which was a simple example where a name and some data was introduced, then this information can be saved and loaded from the mobile phone storage.
- *SquareSequenceAuto*: simple application that reads a preloaded sequence of steps stored into a string with the format:

n_{steps} *V1 V2 V3 V4 PUMP Time LED*

  For example:

1 1 0 1 1 1 09 1 1

  After this, white squares will be shown or hidden depending on the values stored in the string, each step lasts the time determined by the field *Time* in seconds, and once this time passes, next step is executed.
- *VideoCapture* is an app where the camera preview is shown, it is possible to capture video and save it on the phone using a predefined name. Also different aspects related to the video quality and the way the video is stored can be edited. There was also a button for selecting which camera is being used (front facing or back facing camera).

Regarding the video capture, after a lot of hours of work looking for a possible solution about how to process the video recorded allowing a “real time” control of the device, I finally discarded using a video as a source of images, as it was not the most optimum way for reaching our goal. First of all, the videos created occupied big amount of memory, limiting the functionality of the application and also it was not possible to process each frame of the video while was being recorded, so this made the video recording an impossible option.

After discarding videos, next thing that came to my mind was capturing photos, when I told Daniel to try this, he also agreed, since for an optimal control he determined that we needed to process around 5 photos per second, this looked and affordable requirement. I started finding out how to manage the android camera and also looked for different options for capturing photographs. After quite a lot of time working on this part and investigating different possibilities, I found that I was able to capture an image and store it into memory each 4 seconds. After some work I could reduce the time employed for performing this process to 1 second for each photograph, despite this it was not useful for the purpose of the project but it was a better result.

5.2. Manual Process

While I was finding out an optimum way for getting and processing camera frames, Daniel told me the necessity of controlling the device manually for performing some custom tests and adjusting different parameters while the optically controlled application was being developed. As it was a priority task I started with this new application.

It mainly consists in a simplified version of the timed app, where there are four different buttons used for enabling the four different where the fluid will flow through by changing the position of the different valves. When a button is pressed, the two squares corresponded to the pump are hidden for enabling it and also the squares necessary for enabling the corresponding output. When the *STOP* button is pressed, all the squares are shown for stopping the pump and setting the device into the rest mode configuration. If the *STOP* button is pressed again while the device is in rest mode, the app will enable the pump and the last valve selected before pressing *STOP* the first time.

In this mode it is also possible to record video while different configurations are being selected for allowing a later analysis of the test results and the test development.

Figure 7 - Different manual mode states

For programming this application, t programmed different independent blocks and put them together fitting them for my custom purpose. In this case both were used for the timed application and are described in the previous section: *SquareSequence* and *VideoCaptured*

5.2.1.Optically controlled Process

Once I finished the manual application I kept with the image processing, as not so many people need to process live images from camera video and the ones who try to do this often use external libraries like OpenCV (which was not an option for me, as we decided to use the standard libraries and also the documentation provided by OpenCV for Android is not so clear for a novice in the field).

Mainly the idea was using a camera preview external class for showing the images captured by the camera live contained into a Frame Layout, and then get the information of minimum 5 frames per second for processing them.

For achieving this I found out that there was a callback function called *onPreviewFrame* by default, this callback was executed each time a frame was received by the camera preview. My idea was making that callback work and then try to obtain the still image from the camera. As I said, not so many people need to deal with this problem and there was quite few information related to this, most of it was controversial or not useful, but finally I got the solution consulting a forum, in the solution presented, the preview class was declared normally including the preview callbacks enabled, and then in the activity was declared the *onPreviewFrame* callback, which finally worked perfectly.

This was a huge step for continuing developing the application, as achieving this was being the biggest barrier found during the project. With this done I started trying simple things like incrementing a number each time a preview frame arrived and showing it in the phone screen.

When I succeeded in that I started trying to get each image captured by the camera to convert it into a bitmap for its later processing, as this is the easiest way for accessing to the different pixels contained in the images obtained from the preview. For this, first of all is necessary to create a YUV image and convert it to JPEG, it is also necessary to transform that JPEG into a byte array for finally converting it easily into a bitmap.

Testing this application, I found out that it is possible to get 13 successfully bitmaps within a second, more than enough for the specifications imposed by the project. Once the bitmap is available first of all it is needed to start the image processing. For understanding how this process works it is important to know that the goal of the project was being able to fix a known volume of fluid within a selected region of interest.

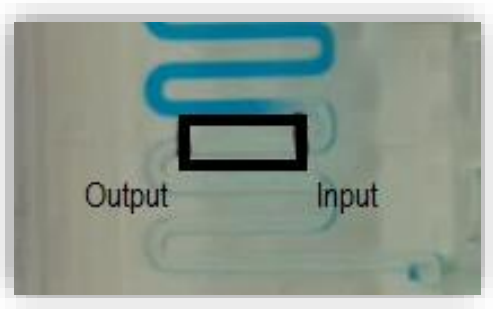

Figure 8 - Defining a region of interest

Next step is detecting when the liquid reaches the input mark and after some time, when this fluid arrives to the output mark it can be said the volume contained between those two marks is determined exactly, as the diameter of the pipe and the separation between the two marks are known, it is possible to calculate the volume of the cylinder.

For an ideal case where the image captured is still and the only thing that moves is the fluid, detecting the two marks (input and output) is quite easy, since the position of the marks is known and will not change over time.

For the first trials I captured a video of the fluid flowing through the pipe and Daniel helped me to split it into a lot of images which gave me. For approaching the problem, we first used Matlab as it is shown in the Annex F for understanding how to do it, and later I started to develop an Android application which would have the same function:

First of all, the image is charged, and then the pixel corresponded to the input mark coordinates is searched, the color of that pixel is compared to a threshold color, if the fluid is not passing through the pipe in that moment, the pixel color detected will be higher than the threshold, and lower for the opposite case. The same happens with the Output mark: the pixel corresponded to the output mark coordinates is searched and then the same color comparison is performed. When both of them are lower than the threshold, this means that the volume delimited between the two marks can be known by measuring the distance between marks since the diameter of the pipe is known by design.

The problem found with method is that in my case, the marks can change its position from one image captured to another if the mobile phone is moved. For solving this, two extra marks were painted in two of the corners of the LOC: the coordinates of one of the corners of the LOC are taken as a reference point, and then the coordinates of the input and output marks are calculated in relative to the corners.

The logic part related to the steps described in the previous page, are contained in a function called *Calibration,* pressing that button black corners are searched and detected. The idea is going through the bitmap matrix and comparing the pixel color to a threshold color corresponded to black. As the mobile phone will be placed more or less at the same position each time (there is a platform where the mobile phone is left for making the tests) it was not necessary to spend computing time scanning the whole image (that situation was also out of the thesis purpose). As a design constraint we decided to scan two isolated areas where I tried to search each painted corner of the LOC as it is shown in the figure.

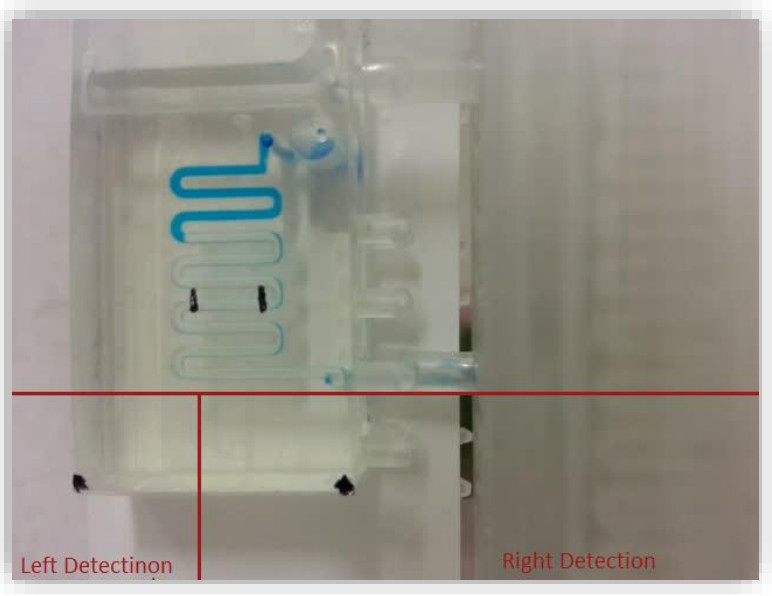

Figure 9 - Detection areas

For this detection it is very important to know that despite the preview shown in the phone screen is 480 x 640 (which is the result of rotating the image received and showing it on the screen), when the bitmap is captured, it gets the real orientation of the camera without any rotation which is 640 x 480.

This is very useful to know, since the way the way the image is traversed depends directly on this and is reflected in the code. To go through the photo, it is started from bottom to top and then from left to right for avoiding conflicts with some darker zones (note that in the image shown in this document there is a white paper below the LOC for facilitate finding the corners as in the end this process is a search of a dark color, then it was important to avoid having extra dark colors in our image captured that may cause false positives.

Once the corners are detected the next step was calculating the two points corresponded to the interest region input and output. For this, it is needed to measure the distance (in millimeters) between one of the LOC corners and the marks corresponding to input and output: containing the part it is going to be observed of the LOC into a plan and taking one of the corners as origin (0,0), the idea is positioning in both x and y axis, the coordinates (in mm) corresponded to the input and the output marks (x1, x2 and y). Also it is important to know which is the distance between the two LOC corners (Δx).

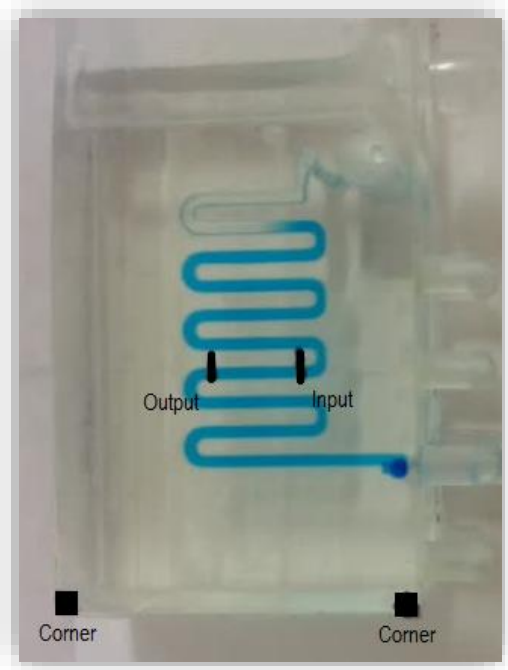

Figure 11 - Points to be detected

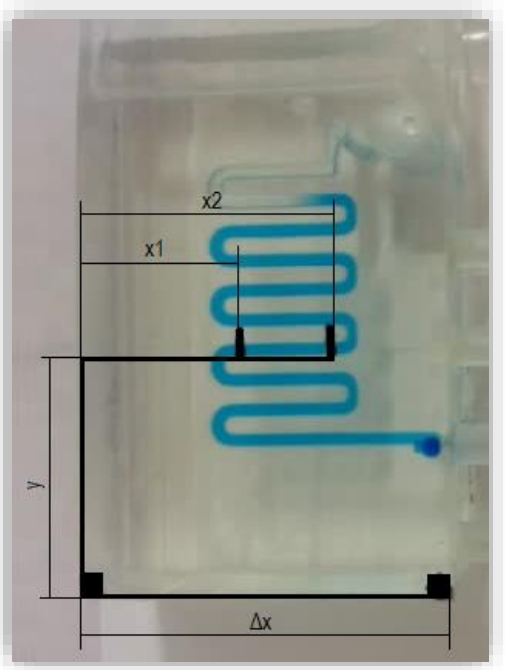

Figure 10 - Measurements needed

Despite the measurements made are in mm, the Android application cannot understand this numbers directly, it is needed to make a conversion between pixels and distance measured. Known Δx, x1, x2 and y, and assuming that the size of a pixel is the same in x and y axis, the pixel density per mm can be calculated as:

$$S_y = \frac{x_f - x_o}{\Delta x} \left[\frac{pix}{mm}\right] \quad assuming\ S_x = S_y$$

Once this density is calculated, can be used as a conversion factor for calculating the coordinates in pixels for the input and the output of the region of interest:

$$x1_{pixels} = S_x \cdot x1\ [pix]$$

$$x2_{pixels} = S_x \cdot x2\ [pix]$$

$$x3_{pixels} = S_x \cdot x3\ [pix]$$

After calculating the equivalent in pixels the next thing was checking the color of the pixels located in that positions and controlling: after the calibration, when the program is running and nothing happened before, the output S1 is open, and the pump is pumping fluid, if a color below the threshold color is detected in the input mark, S1 is turned on and off during a variable period of time which can be adjusted by the user, and finally when the liquid reaches the output mark, the pump is deactivated, achieving our goal which was fixing a concrete volume between the input and the output.

The layout of the application is shown in the *Figure 11,* when the button calibration is pressed, all the process described in this section starts working once and the processing part is executed each frame is received, and when stop is pressed, the process is halted until calibration is clicked again. Also the color of the pixels which delimit the region of interest color two squares in the upper part of the application, overwriting the Up color and Down color labels.

Figure 12 - Optically controlled app

For programming this last application I also made some independent blocks for checking that all parts worked before combining them together for controlling optically the device:

- *DoActionWhenReceiveOneFrame:* this application opens the camera preview and each time a frame arrives, a variable is incremented and printed in the screen.
- *Calibration:* this app opens an image preloaded in the drawable folder as a bitmap and starts searching the corners of the LOC as it has been described during this section, finally the app returns the coordinates of the two points corresponded to the input and the output of the region of interest.
- *DetectColorsInDeterminedPixel:* this application is equivalent to the application shown in the Annex F but programmed for Android. Using a pre-loaded image sequence the color is detected in two fixed points (corresponded to the input and the output of the region of interest) of each photograph. The app can detect when the fluid has entered into the region of interest, and when it reaches the output mark.
- *TouchColorDetection:* an image is shown on the screen; this application detects the color of a pixel that the user has pressed with the finger. Both the color and the coordinates where the pixel can be found are shown on the screen.
- *DetectPointsInRealTime:* each time an image frame arrives, the calibration is executed, this way it can be said that there is a continuous detection of the points, and then there is a real time knowledge of where is the region of interest.

6. Conclusion and future work

Finally, the project came to its end. Before closing this document, it may be relevant to evaluate the fulfilment of the goals that were set prior to the start of this work

- Redesign of the platform for adapting it to the needs imposed by the new design of the application, learning that small modifications in a device can produce great results. Discovered the versatility of 3D printers and the great possibilities in terms of design and prototyping.
- Understanding the previous application, redesigning, and adding some new functions for the upcoming changes. From the total ignorance about Android, a solid theoretical and practical programming base was acquired while the application was being developed.
- Design of the feedback stage and controlling software. Measurement carried out on a LOC, created with the fast prototyping unibody-LOC (ULOC15) technique developed at ODL. Understanding of image processing and computer vision techniques adapted to Android programming without the use of external libraries.
- Writing the Project Document as a result of the accomplishment of the mentioned goals, this detailed document mainly sums up all the knowledge acquired during the development of this project.

6.1 Future work

As this project was part of a bigger one, where the whole ODL investigation group was working on, the next steps were first of all, make some test to adjust the operation of the application and add new functionalities if necessary.

Some of the functionalities of the software designed could be improved are the following:

- Implementing the detection algorithm for having more than one region of interest was one of the future goals to achieve for finally controlling the process, this remaining work would be continued by Germán for continuously develop a valid application adequate to the needs of the tests.
- Implementing a better control of the times introduced by the user in the timed application.
- Combining the three different applications into a single one with a menu which allows selecting between the three options.
- Improving the graphic part and adapting it to different phone screens.

Referencies

Alicante, U. (2016). *Experto Java Universidad Alicante*. Obtenido de http://www.jtech.ua.es/dadm/restringido/android/sesion03-apuntes.pdf

Alicante, U. d. (2012). *Experto Java*. Obtenido de http://www.jtech.ua.es/dadm/restringido/android/sesion03-apuntes.pdf

API, A. D. (2016). *Android Development Web*. Obtenido de Android Development Web: https://developer.android.com/guide/topics/media/camera.html#camera-preview

Comina, G., Suska, A., & Filippini, D. (2015). Towards autonomus lab-on-a-chip devices for cell phone biosensing. *ScienceDirect*.

Dimitri. (21 de Abril de 2011). *41 POST*. Obtenido de http://www.41post.com/3719/programming/android-how-to-return-rgb-values-from-an-image-file

Preechaburana, P., Suska, A., & Filippini, D. (2014). Biosensing with cell phones. *Cell Press*.

Profesorado, I. N. (s.f.). *educaLab*. Obtenido de educaLab: http://platea.pntic.mec.es/~lgonzale/tic/imagen/conceptos.html

Stack Overflow. (s.f.). Obtenido de http://stackoverflow.com

Wikipedia.org. (2016). Obtenido de Wikipedia.org: https://es.wikipedia.org/wiki/ELISA