Álvaro Bolaños Rodríguez

IR camera in Well-being technology

Creating a cloud based communication channel

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

24.4.2017

Metropolia

Author(s)	Álvaro Bolaños Rodríguez
Title	IR camera in Well-being technology
Number of Pages	47 pages + 3 appendices
Date	Monday 24th April, 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialization option	Software Engineer
Instructor(s)	Dr. Tero Nurminen, TKI-vastaava yliopettaja

The objective of this thesis was to develop an effective bidirectional communication system for IR sensors from the company LeViteZer using an IoT approach and connecting them through cloud services and mobile networks provided by the Nokia Innovation Platform.

On the thesis there is described different methods to build this communication channel from a full stack point of view. In this case there are a sensor-side, a client-side and a server-side methods.

The outcome of this project was an set of pieces of applications to make the communication system using technologies and frameworks available to any software developer. This communication system as the date of the publication of this thesis was tested and met the expectations set on business challenge on the Introduction chapter and it is likely to be continued its development.

It is discussed possible applications of this system in the context of the Nokia Innovation Platform, Well-being services and some other cases.

Keywords	IoT, IR, sensor, cloud, full-stack, communication channel, websocket, http, LTE, Android, Python

Contents

Abbreviations and Terms

3G Third Generation Mobile Network.

API Application Program Interface.

C++ C plus plus.
CPU Central Process Unit.
CVS Control Version System.

ER Diagram Entity Relationship Diagram.

HTML HyperText Markup Language.
HTTP Hypertext Transfer Protocol.
HTTPS HTTP over SSL.

I/O Input/Output.
IaaS Infrastructure As A Service.
IoT Internet of Things.
IP Internet Protocol.
IR Infrared.

JSON JavaScript Object Notation.

LTE Long Term Evolution.

NAT Network Address Translation.
NPM Node Packet Manager.

OOP Object Oriented Programming.
OTG USB On-The-Go.

PaaS Platform As A Service.
PDF Portable Document Format.
PHP Hypertext Preprocessor.

RAM Random Access Memory.
RAN Radio Access Network.
RTP Real Time Transport Protocol.

S2I Source-to-Image.
SaaS Software As A Service.
SIM subscriber identity module.
SPI Serial Peripheral Interface.
SQL Structured Query Language.
SSH Secure Shell.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.
UI User Interface.

UML	Unified Modeling Language.
URI	Uniform Resource Identifier.
USB	Universal Serial Bus.
VoIp	Voice over IP.
VPN	Virtual Private Network.
WWW	World Wide Web.
XML	Extensible Markup Language.

Android	Operative System used mainly in Smart-phones.
Apache	the most used web server software in the world.
Docker	Software container platform.
Git	Control version software.
GitHub	Hosting site for Git repositories.
Java	Programming language able to run on most of the Operative Systems.
Java ME	Java Micro Edition: Java version for mobile or embedded devices.
Javascript	Dynamic programming language used mostly in browsers although can be used in desktop and server applications..
Kubernetes	Automated container deployment, scaling, and management.
LabVIEW	environment for visual programming language used for instrument control.
Matplotlib	Plotting library for Python programming language.
MIDlet	A MIDlet is an application that uses the Mobile Information Device Profile (MIDP) on Java ME environment.
Node.js	Node.js is an open-source, cross-platform JavaScript runtime environment for developing server-side applications.
Python	Dynamic typed interpreted programming language.
Qt	Cross-platform application framework.
RS-232	A standard for serial communications.
TCP/IP	All the necessary layers to match the conceptual model of the Internet protocol suite. .
Websocket	Application protocol build on top of TCP.

1 Introduction

Thermal images have a number of advantages over conventional light-based video camera images. These thermal images can tell not only whether there are living people or animals but also whether there is any temperature anomaly on them. They can be used to make assumptions about their physical state or understand how a group of individuals move or behave using Computer Vision software.

Keeping into account that Nordic European countries and specially Finland are rapidly aging, costs in elder and general health care are becoming more expensive. Health care entities may decrease their costs by search for new ideas in the field of IT. In the discussion chapter some ideas to use Infrared (IR) sensors and the communication channel are explained [1].

Here is where Internet of Things (IoT) comes into play, Nowadays the Internet is very accessible and fast. Almost every conceivable device: phones, watches, televisions, speakers, cameras, etc can be connected, send and receive continuously big amounts of data.

Although with IoT it comes the process of defining a communication channel between the IoT devices, services, clients, databases, etc. In order to manipulate, visualize, store and distribute data from them. On this thesis it is explained the steps that were followed to develop the solutions to these topics and what alternatives were tried and what tools/technologies were used to carry out the project.

Background of the case company

The thesis originates in a collaboration with the case company called LeViteZer (http://www.levitezer.com/), Metropolia and Nokia. LeViteZer is company that develops controllers for cameras for image stabilization such as gimbals [2]. They provide the IR sensors. Nokia with their innovation platform [3], provides the cloud environment and the

network. My part as Metropolia student is develop the mentioned communication channel.

Business Challenge

The case company wants to find a system which uses IR sensors that allow us to monitor patients while keeping their privacy along with reducing costs by using less staff and improve the service quality. Moreover, it should be possible to access data from those sensors fro anywhere which make them very versatile and portable.

Objective and Outcome of the Study

With the business challenge in mind, this study aims to answer the following question:

> How to create an effective communication channel between ir sensors and clients such as computers, laptops, etc and use it on Well-being services?

The outcome of this study is:

1. Different client-side applications able to connect to sensors.
2. An web Application Program Interface (API) that connect IR sensors to a cloud hence to clients.
3. Proposals to use the resulting communication system on health care institutions.

2 Theoretical Background

Communications between devices is not a new topic, there are plenty of methods and communications protocols, this can be also part of the problem: there are too many of them and sometimes this can be overwhelming. In this section it is described what is known about communication and network protocols in order to find the best way to create a communication channel as stated in the business challenge on section 1 of the introduction.

2.1 Communication between devices

In order to establish a communication between two ends in a computer network or in Internet it is good to comprehend how the ”Internet Protocol Architecture” or TCP/IP stack works. Most of networks are based on it, including office and home networks [4, 9].

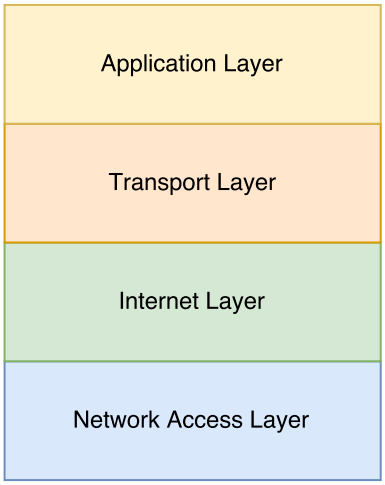


Figure 1: Transmission Control Protocol (TCP)/Internet Protocol (IP) Architecture

The TCP/IP stack is build in layers:

1. Network Access Layer: physical medium to access the network.
2. Internet Layer: handles the routing of data
3. Transport Layer: provides host to host data delivery services.
4. Application Layer: applications and process that make use of the network

Then, a way to communicate through the TCP/IP stack as shown in figure 1 must be found.

2.2 The medium

At this point it has to be decided what physical medium to *access the network* (Network Access Layer in figure 1). It could be a simple copper cable, Wi-Fi, microwaves, laser, etc. almost any kind of electromagnetic wave. But since this is an IoT project the best approach will be using Radio Access Network (RAN) which it is accessible from cell antennas and nowadays provide great speeds and bandwidth.

As stated in the Introduction, Nokia provides access to a mobile network. This network is called NetLeap which uses Third Generation Mobile Network (3G) and Long Term Evolution (LTE) technologies (the same that use mobile phones to connect to Internet). NetLeap is a closed network for research managed by Aalto University and Nokia [5].

2.3 Network protocols

The Internet Layer protocols usually rely in routers and other apparatus which are out of our control, hence the only concern is about transport and application layer.

The idea is to use reliable application protocol to make the connections and this should be platform independent and a Internet standard, these protocol standards specifications are available officially in https://www.rfc-editor.org/standards. On the next sections are the protocols considered and the reasoning behind them.

2.3.1 UDP

User Datagram Protocol (UDP) is a connectionless transport protocol, it does not guaranty delivery nor order of packets which means they can get lost and will not be re-requested and might come in a different order than when they were sent [4, 18].

UDP is commonly used to provide real communication such as time video stream on protocols as Real Time Transport Protocol (RTP), it is also used in Voice over IP (VoIp) to deliver telephone calls over network. They take advantage about the connectionless nature of UDP which despite the mentioned disadvantages it has a low latency. On the other hand losing some packets during a call or video retransmission is not a big deal.

Then a UDP communication system may be suitable for this project since, what it is sending from the IR sensors is a binary stream of images. Commands can be sent over UDP as well. This can be done sending packets between sensors and clients directly or through a server which could coordinate the data flow.

Port issues on remote hosts

Routers and firewalls usually do not accept connection from ports other than 80 and 443. This is an issue when using UDP (or TCP) sockets approaches. In a local network usually there are no such issues (although firewalls could be strict, depends upon the local network administrators), so a solution can be using a Virtual Private Network (VPN) provider which allows remote computers act as if they where in a local network and increase the security as well.

Another solution is UDP hole punching technique to establish bidirectional communication between hosts that are behind Network Address Translation (NAT) routers by using a external host to keep track of the ports and addresses used in the NAT tables of both hosts routers.

2.3.2 TCP

TCP is the other transport layer protocol, unlike UDP, TCP is connection based and guaranties the delivery and order of packets. Thus protocols made on top of TCP establish a connection and have to maintain it, which increases the latency [4, 19].

Http

Hypertext Transfer Protocol (HTTP) and HTTP over SSL (HTTPS) is application layer protocol made on top of TCP. HTTP is meant to request resources such as HyperText Markup Language (HTML) documents, Extensible Markup Language (XML), JavaScript Object Notation (JSON) or plain text. Any request has a response which can contain a body of data and response code (like the famous ”404 not found”) [6].

HTTP is strongly related to the World Wide Web (WWW), on most of routers and firewalls the port 80 (HTTP) and 443 (HTTPS) are allowed. In addiction, the HTTP responses can be used to receive data from the other end indirectly. This could be used for a request-response sensor-client communication approach using a third party like a server.

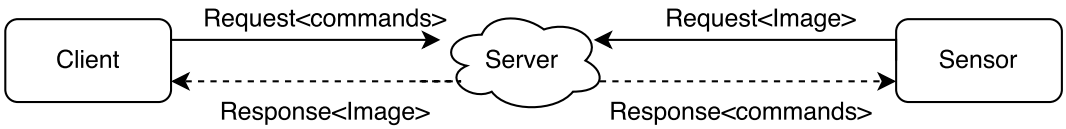


Figure 2: Http/s approach

As seen in figure 2 both sensor and client use requests to send data (which is between ”<>” signs in the figure) that other end expect to receive in the response. A server application take cares to forward requests data to responses. Thus all requests are always towards the server and must be continuous.

Websocket

Websocket is a relatively new protocol (end of 2011) which was meant to provide web applications with bidirectional communication without making continuous HTTP requests through techniques like XMLHttpRequest [7, 4].

As HTTP, Websocket use ports 80 and 443 (secure Websocket) by default then it can go through NAT and firewalls easily. Once a connection is established both ends can receive and send data until the connection is closed. This connection is started with a handshake in form of a HTTP GET request [7, 6].

3 Methods and Materials

The aim of this project is to design methods to transmit data from one sensor to a client application and vice-versa. Also define a generalization to communicate from N sensors to M client applications, being either N and M arbitrary numbers. Note that the communication must be bidirectional since clients can send commands to sensors in order to perform operations such as calibration, delay between frames, etc. On this communication system there are 3 well differentiated parts:

- Sensor side: software that connects the sensor with the server side.
- Server side: software that connects sensors and clients together.
- client side: software that connects the an user with the server side to access a sensor.

The figure 3 shows this idea.

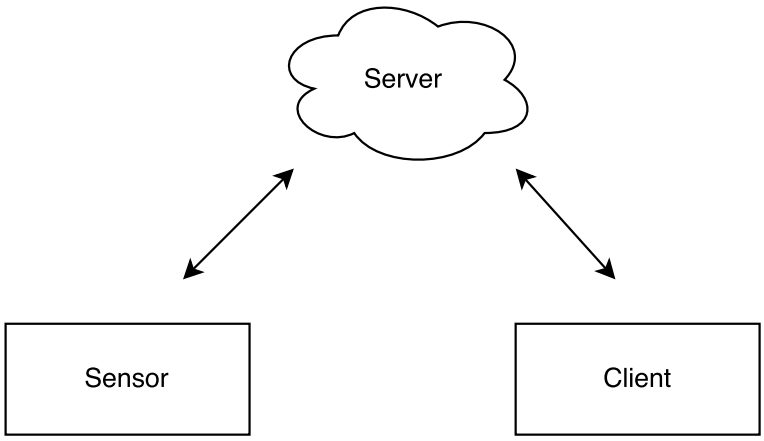


Figure 3: A communication channel

This chapter discuss about methods for each side of the communication system in detail.

3.1 The sensor

The IR sensor provided by LeViteZer delivers all infrared data in form of binary streams trough Universal Serial Bus (USB).

In order to make the an image it is necessary to process those streams. Every image or frame comes in 240 rows of 80 bytes of data separated by a delimiter of 3 bytes plus an extra byte that identify the row:

$$\text{FF FF FF 00}\,\{data\}\,\text{FF FF FF 01}\,\{data\}\,\text{FF FF FF 02}\,\{data\}\ldots \tag{1}$$

In equation 1 every byte is on hexadecimal format containing a sequence ”FF FF FF” which is at the beginning of every row. After this sequence, the 4th byte is the number which identify the row from 0 to 240.

As equation 2 shows, The 240th (F0 in hexadecimal) and last row provides meta-data about the frame

$$\ldots\text{FF FF FF F0}\,\{metadata\} \tag{2}$$

In table 1 are listed all information contained in the meta-data.

Meta-Data

Every frame comes with valuable data about its state and sensor’s such as configuration, temperature parameters, etc.

Meta-Data Parameter	**Bytes (from 0 to 80)**
Time counter	4,3,1,0
Frame counter	10,9,7,6
Frame Mean	13,12
Sensor temperature	16,15
Maximum temperature	19,18
Minimum temperature	22,21
Discarded packets count	25,24
Maximum temperature limit	28,27
Minimum temperature limit	31,30
AGC byte	34
Bit depth	35
Delay between frames	37,36

Table 1: Meta-data and its position in the row

Here is a short explanation about the meta-data values:

- Time counter: Amount of seconds since the sensor was power on.
- Frame counter: Amount of frames since the sensor was power on.
- Frame mean: Temperature mean of the frame.
- Sensor temperature: temperature of the sensor itself.
- Maximum temperature: maximum temperature registered the current frame.
- Minimum temperature: minimum temperature registered the current frame.
- Discarded packets: packets that were not read. a great number may tell that the application is not reading the sensor fast enough.
- Maximum temperature limit: The limit set with the command for maximum limit.
- Minimum temperature limit: same that previous but with minimum temperature.
- AGC byte: tell if the limits are set or not (useful for implementing indicators).
- Bit depth: bit depth of the image, it can be 0, 2 or 8 (default).
- Delay between frames: if no delay is set (delay=0) then the delay is about 111 milliseconds (9 frames per second).

Note: Temperatures from sensors are not in absolute values, this is because the sensor does not detect particular values but difference in temperature.

Commands

The commands are sent over the same serial USB cable from which the frames are received. In table 2 the main commands are displayed along its binary representation: for the first byte an American Standard Code for Information Interchange (ASCII) character and for the data argument it depends upon the command.

Command Name	Command (ASCII byte)	Arguments
Synchronize	S	No
Calibrate	C	No
Set Maximum Temperature Limit	H	2 bytes
Set Minimum Temperature Limit	L	2 bytes
Auto Maximum Temperature Limit	A	No
Auto Minimum Temperature Limit	a	No
Set Bit Depth	B	1 byte
Set Frame Delay	U	2 bytes

Table 2: Commands accepted by the sensor

Note: The frame-rate is 9 frames per second at maximum, although the sensor can be configured with an arbitrary delay time between frames.

Creating the image

Every frame has a size of 160x120 pixels, but the image data comes in a matrix of 80x239 (240 is the meta-data row) as equation 1 and 2 shown before. Each byte of data can be seen as a pixel in a gray-scale image, but in order to generate the correct image the ”data matrix” must be reshaped to 160x120 as show in the figure 4. Every 2 row in the data matrix make 1 row in the image.

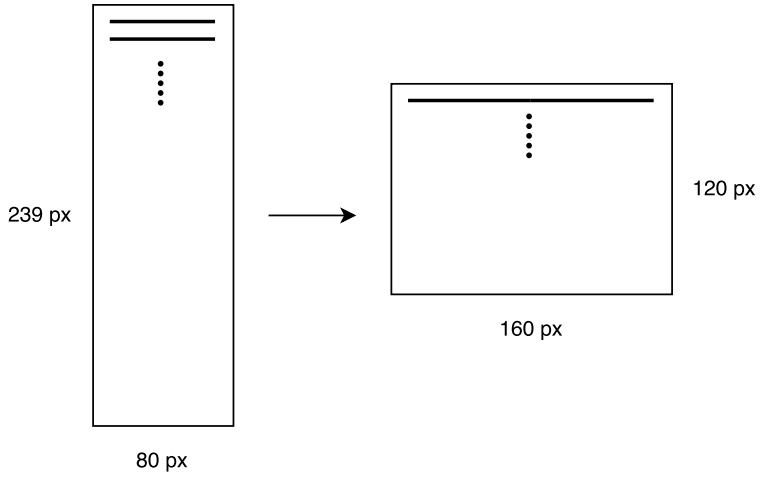


Figure 4: Reshaping data

In the future the number of pixels might change so it is better to define a generalized solution to do the reshaping:

For each data row (equation 3) let n_j be the data current row number performing by the 4th byte in equation 1, then for every d_i value in the data row is possible to define every pixel p_{ij} of the image matrix (equation 4) as a coordinate pair (x, y) in the data matrix. s is the size of the data row which is 80 in this case.

$$D_{i,j} = \begin{pmatrix} n_1 & d_{1,1} & d_{1,2} & \cdots & d_{i,1} \\ n_2 & d_{2,1} & d_{2,2} & \cdots & d_{i,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n_j & d_{1,j} & d_{2,j} & \cdots & d_{i,j} \end{pmatrix} \tag{3}$$

$$I_{i,j} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,j} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ p_{i,1} & p_{i,2} & \cdots & p_{i,j} \end{pmatrix} \tag{4}$$

$$x = n_j \backslash 2 \qquad y = n_j \bmod s + i \tag{5}$$

$$p_{ij} = d_{xy} \tag{6}$$

As seen in equation 6 any pixel value corresponds to a x,y pair define in equation 5 .

Note the ”\” here is meant for integer division in equation 5, it is **not** a normal division with rational or decimal numbers. ”mod” function represents the modulus operation which finds the remainder of a division.

As an example of a practical implementation in Python see the listing 1 which function process_data_row is called for every data row

```
1 self.frame_arr[f_row][f_col]
2
3 def process_data_row(self, row):
4     n_row = row[0]
5
6     for indx, val in enumerate(row[1:]):
7         f_row = (n_row)/2
8         f_col = (n_row) % 2 * 80 + indx
9         self.frame_arr[f_row][f_col] = val
```

Listing 1: Simplified example of creating a frame in Python

Also note that in order to fill the ”self.frame_arr” matrix the function is must be called 239 times

3.2 Sensor Side Methods

The sensor has been described in the previous section. Here different methods to read the sensor are discussed, some worked better than others, nevertheless all of what was tried is included.

3.2.1 Android smart-phone as Gateway

Most of Android smart-phones have an USB On-The-Go (OTG) which allows to use USB peripherals in the phone with the correspondent OTG adapter This can be used to develop an Android application to receive the data from the sensor and send it through Internet using any protocol, in this sense the cellphone act as a gateway to Internet, letting the sensor access the 3G or the LTE network.

An advantage of this approach is that the smart-phone(including a subscriber identity module (SIM) card) has all the hardware on it to make the communication. Hence there is only need to focus on the software part.

This was tested using a Hypertext Preprocessor (PHP) file in a server on Metropolia UAS alongside an Android Application I developed to send all the data from the sensor to server.

Note that this method requires a terminal which supports USB OTG.

Description of the Android Application

To create the Android application it is needed some things:

- A minimal User Interface (UI)
- A network protocol and its implementation
- A background process

The UI let us start the reading and provide an address and a port to connect. See figure 5.

Figure 5: UI of the Android Application

As network protocol HTTP was chosen, although it is not the best for continuously sending data. Then an PHP script processed the request and store the data in form of hexadecimal strings in a file as listing 2 illustrates

```
1  <?php
2  $data = "";
3  // get data only from post request
4  if ($_SERVER["REQUEST_METHOD"] == "POST") {
5      $data = ($_POST["data"]);
6  }
7  // write it to file
8  $fileName = "data.txt";
9  $file = fopen($fileName, "a");
10 if($file){
11     fwrite($file, formatData($data).",");
12     fclose($file);
13     echo "OK";
14 }
15
16 function formatData($data){
17     $arr = unpack('H*', $data);
18     return strtoupper(implode(" ", $arr));
19 }
20 ?>
```

Listing 2: PHP code in the server side

Then using an Android service to keep reading in the background which is the usual approach to deal with long running operations on Android Applications [8].

The usb-serial-for-android library provides all the necessary to read data from the mini-usb port [9]. The listing 3 shows an extract of my code implementing the callback to receive data in directly from the USB port.

```
1  // the parameter data is a binary string from the sensor
2  @Override
3  public void onNewData(final byte[] data) {
4      if(bufferFrames.isFull()){
5          callback.getBuffer(bufferFrames);
6          bufferFrames = new BufferFrames();
7      }else{
8          bufferFrames.addChunk(new Chunk(data));
9      }
10 }
```

Listing 3: Android: Callback to receive data from sensor

Notice in line 3 that it is in reality quite simple to receive binary data in form of an byte array.

3.2.2 Lte module

Long Term Evolution (LTE) modules work as a phone: They need a SIM card to connect to network, they can be integrated in a board. Typically these modules have a number of interfaces such as USB, rs-232, Serial Peripheral Interface (SPI), etc to connect peripherals.

In order to set a route between your device and a service on Internet or your own server, a piece of code must be provided, it depends upon the module how can be done. For a project like this it is interesting that the module has its own TCP/IP.

Gemalto Lte module

This is not a simple modem that allows other machines to be connected to Internet, it has a complete TCP/IP stack which means that protocols of the transport layer such as TCP and UDP and application layer such as HTTP and HTTPS can be used as well all in very small compact chip as it is seen in figure 6.

The module used was Gemalto ELS61-E which is configured using the Hayes command set (also called AT commands) which are used typically on modems. See how AT commands look in listing 4.

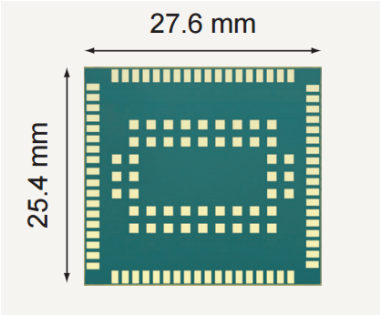


Figure 6: ELS61-E chip size

```
1 AT^SMSO  # shutdown
2 AT+COPS  # register to network command
```

Listing 4: AT commands

This module can be programmed using Java ME which is a Java edition for embedded devices [10]. Application layer protocols and TCP/UDP sockets can be used in a MIDlet which allows to describe the life-cycle of a single application such as in the listing 5 example. With this tools the USB, SPI or RS-232 ports can be controlled and even send AT commands.
For a full list of features of the module refer to http://www.gemalto.com/brochures-site/download-site/Documents/M2M_ELS61_datasheet.pdf

To do all the communication and configuration with the module there is a board where it can be attached proving micro-usb connectors, antenna, reset button, power among other things see figure 7

3.2.3 Raspberry Pi

A Raspberry pi is a credit card sized computer and on its model 3B includes among other things 4 USB ports multi-core Central Process Unit (CPU) [11]. Considering that the raspberry comes with a Linux distribution the possibilities are unlimited, for the purposes of this project it is specially handy the possibility of using any programming language and a lot of software packets available.

```java
import javax.microedition.midlet.*;

public class HelloWorld extends MIDlet {

    public HelloWorld() {
        System.out.println("Constructor");
    }

    /** This is the main application entry point. */
    public void startApp() throws MIDletStateChangeException {
        System.out.println("startApp");
        System.out.println("\nHello World\n");
        destroyApp(true);
    }

    /**  Called when the application has to be temporary paused.
       */
    public void pauseApp() {
        System.out.println("pauseApp()");
    }

    /** Here you must clean up everything not handled by the
       garbage collector. */
    public void destroyApp(boolean cond) {
        System.out.println("destroyApp(" + cond + ")");
        notifyDestroyed();
    }
}
```

Listing 5: MIDlet application life-cycle example

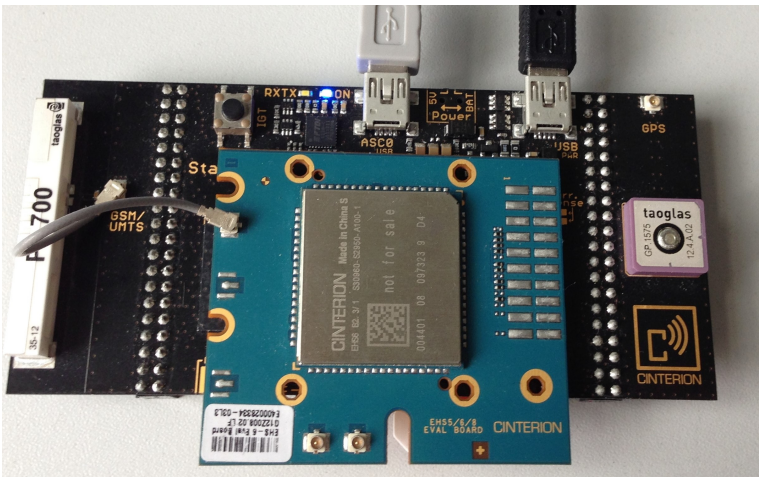


Figure 7: Lte Module on top of the board to program it

The reader program

The programming language chosen to communicate with the sensor from the raspberry pi was Python since it has a number of libraries available to use. Then it is necessary to point

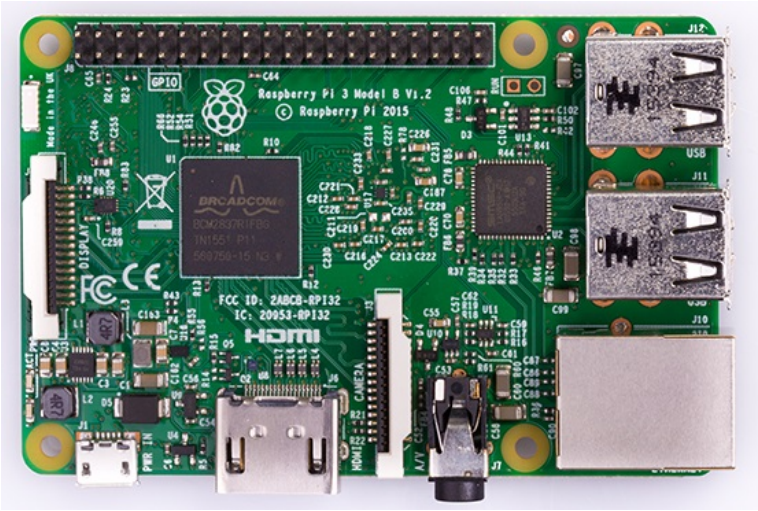


Figure 8: Raspberry Pi Physical appearance

what features are suitable for a program which reads the sensor and open a connection to the server side:

- Needs to communicate both ways trough USB.
- Has to open a connection to the server/cloud continuously and be able to recover. automatically from failures on the network.
- Must do all this at same time taking advance of raspberry pi's multiple cores.
- Prioritize the sensor read task over the rest, this is the most critical part and must be have a higher priority

In order to separate tasks it is possible to use multiple threads which are supported in Python and although this allows run several tasks at the same time when using python, it could be advisable to use the multiprocessing package to separate tasks into several process and take advantage of the various CPU cores [12]. For controlling process priority the package ”psutil” allows to control the ”niceness” of the process [13].

In order to get access to the raspberry when it is out of reach, connections to it through Internet using Secure Shell (SSH), which is a protocol to connect to remote hosts using shell through an encrypted connection. It is also desirable being able to do certain operations remotely without using SSH. The connection made between client-server-raspberry can be taken in advance to for example shutdown, reset, update, etc the raspberry. For this purpose the subprocess package can be use to issue commands and other process, as one can do in a terminal shell [14]. See the listing 12 on appendix 1 to see my actual implementation.

Services

To run programs and script automatically from booting it is possible to add systemd based services which can be found in most of Linux distributions. For the purposes of this sensor reader it is needed at least two services. One to keep alive the network connection alive. Other to keep the reader program constantly running. Listing 6 show these two services which basically run other script and try to keep it alive.

```
1  ### /lib/systemd/system/lte.service
2  [Unit]
3  Description=Lte module service
4
5  [Service]
6  ExecStartPre=ls /dev/cdc-wdm0 || echo "cannot see the lte module,
       retrying..."
7  ExecStart=/home/pi/lte-daemon
8  Restart=on-failure
9  RestartSec=5
10
11 [Install]
12 WantedBy=multi-user.target
13
14 ### /lib/systemd/system/sensor.service
15 [Unit]
16 Description=read sensor and send data via websocket
17
18 [Service]
19 ExecStart=/home/pi/sensor-reader/main.py
20 WorkingDirectory=/home/pi/sensor-reader
21 Restart=always
22 RestartSec=5
23
24 [Install]
25 WantedBy=multi-user.target
```

Listing 6: Systemd services

3.3 Client Side Methods

On this section it will be described different methods to create a client able to connect to the cloud thus the sensor, present the data to the user and send commands to the sensor.

Porting the LabVIEW Application

At the beginning of the project LeViteZer provided an example application to read the sensor from a laptop made in LabVIEW, although this works it is not an idea platform to develop since it is strict closed software and in order to work with it a expensive license has to be paid. Then it was agreed that a ported python version will be made from the LabVIEW client and extends its capabilities beyond the laptop-sensor USB connection.

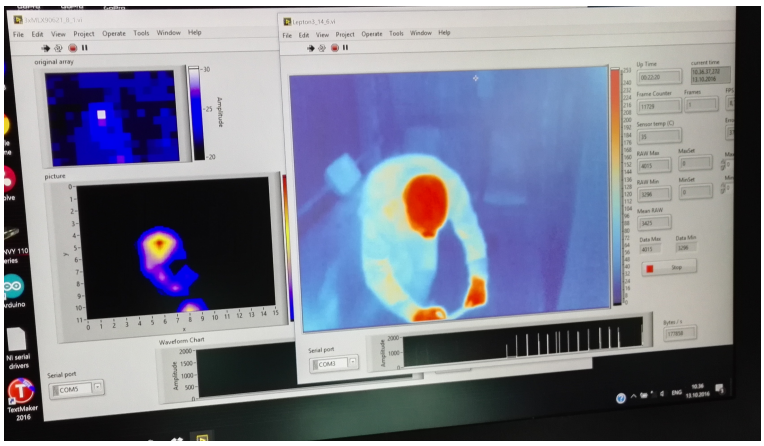


Figure 9: LabView implementation

LabVIEW programs are not write in code but using a ”visual programming language” based in diagrams similar to circuits were the data flows. It is oriented to instrument control, data acquisition, automation, etc. [15]

Although there was not intention to use LabVIEW for the purposes of this thesis, it does was used to do research about what protocols to use and testing, see figure 10 for a simple UDP client applications which just receives raw binary information from a sensor and displays it alongside the frame number.

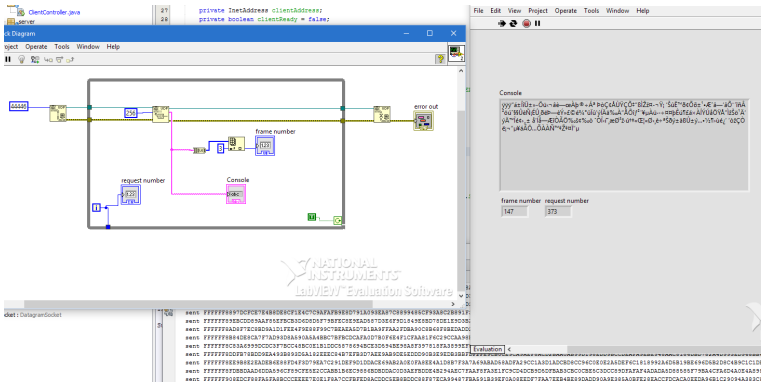


Figure 10: Labview UDP client

3.3.1 Websocket Python Client

This application was intended to have the characteristics of the LabVIEW's mentioned on the previous section and provide it with connectivity to a cloud service for remote communication. The application was wrote on python and taking advantage of the multiple libraries that the community offers for free as open source.

Design

On a relatively complex application as this one Object Oriented Programming (OOP) is the most convenient way to go when designing the application.

As data sources in this application, it can be from directly USB or from network, for those purposes the class "SerialConnection" will be in charge of communicate with a IR sensor connected to the computer directly over USB (for testing) see listing 10 in appendix 1, this class is the same that is used in the raspberry side to read the sensor. On the other hand "WebsocketConnection" mimics roughly the behavior of "SerialConnection" however it makes the connection over Internet using the Websocket protocol.

Once a connection is made the class "Camera" and its children will handle data processing to make array frames as it was explained in section 3.1.

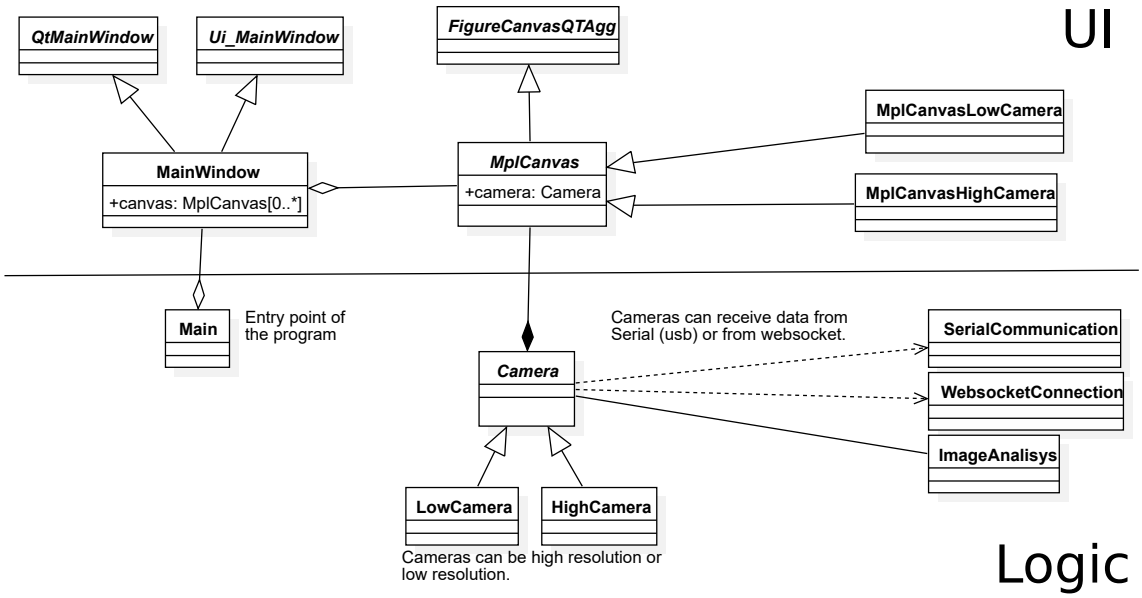


Figure 11: UML diagram (simplified)

As seen in figure 11 it shows the most important components of the application and its relations, it can be noticed that there is a separation between UI components and logical ones.

User Interface

The user interface is implemented using the known Qt framework which is uses C plus plus (C++) but there is a python bindings package to use in a python application without writing a single line of C++ code [16].

Advantages of using qt are among others:

- cross-platform: same code works on any operative system where the framework is available.
- it is possible to design the interface using a designer program (see figure 12) and save it as a XML file that can be read from the application, saving a lot of time on development stage.
- it is a well known framework and there is plenty of information available about it.

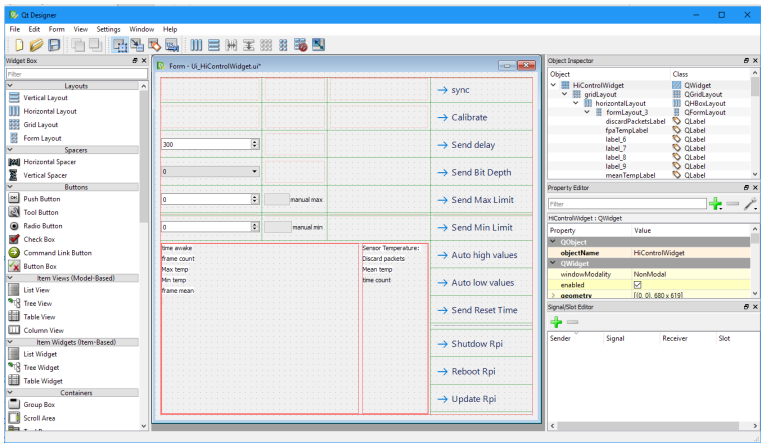


Figure 12: QT designer

The sensor image itself is made using a plotting library named Matplotlib used in quality scientific plots and animations. Matplotlib also provides a back-end to attach the graphics to Qt among other UI frameworks. This can be seen in the Unified Modeling Language (UML) diagram (figure 11) where the class ”MplCanvas” which presents the frames from the ”Camera” inherits from the Matplotlib class ”FigureCanvasQTAgg” [17].

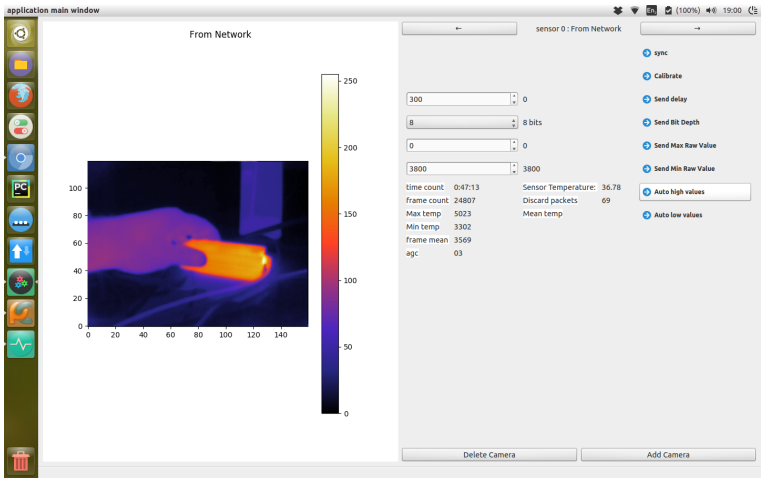


Figure 13: Python Client Application

As seen in figure 13 the application client has 2 well differenced parts. On the left it is the image displayed as a colored gray-scale image in figure 14. Matplotlib let us apply color maps on the image very easily, also to add an interpolation to improve the image quality, in this case there is a Bicubic interpolation [18].

Figure 14: Ir image

On the left side there is a control panel (figure 15) which displays meta-data information and buttons that can send commands to the sensor as described in section 3.1, on the right of some buttons there are input fields to enter the arguments to the commands that required it alongside the current value of the argument (current argument values are meta-data as well).

There is a separate section for special buttons to control a raspberry pi in the case the sensor is connected to a one.

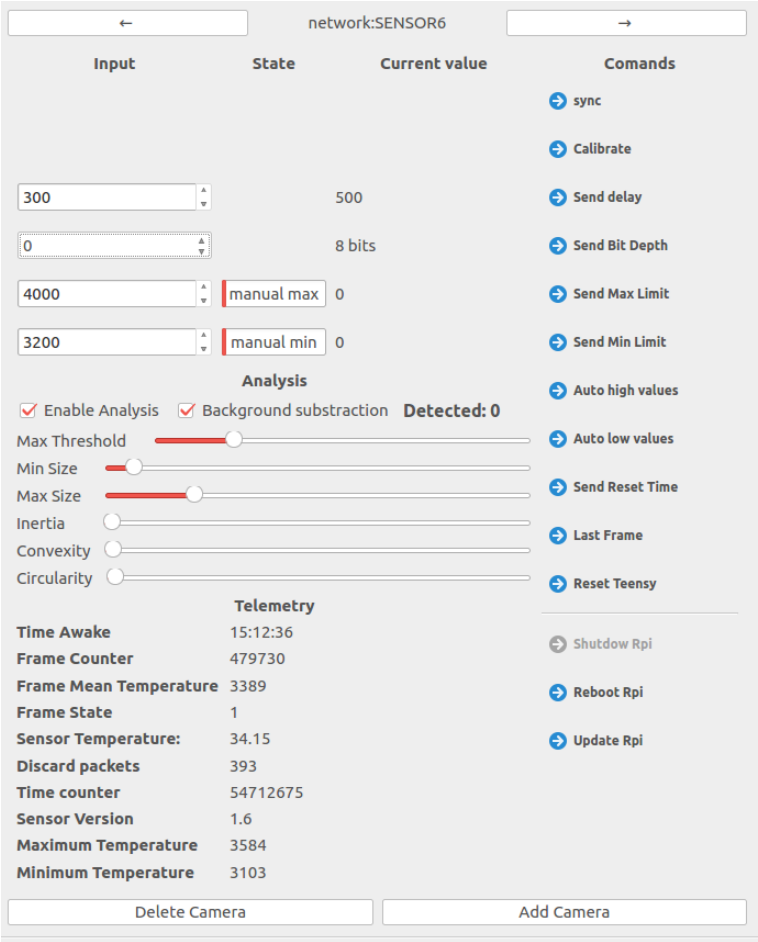


Figure 15: Control panel

3.3.2 Android Client

During the development of the project I found attractive the idea of having a client application on the phone and after finishing the python client described in the previous section I started to develop a simple but powerful Android application. It let connect to any camera already registered in the cloud (At this stage the cloud allows multiple sensors see. section 3.4).

As in the python client, it is necessary to code:

- a class to connect to server or cloud through Websocket.
- create the image from the data.
- allow to choose which camera to connect to.

Android application are slitted on different screens called Activities which contain UI ele-

ments that the user can interact. In this case there is an activity for choosing the camera to connect, see figure 16.

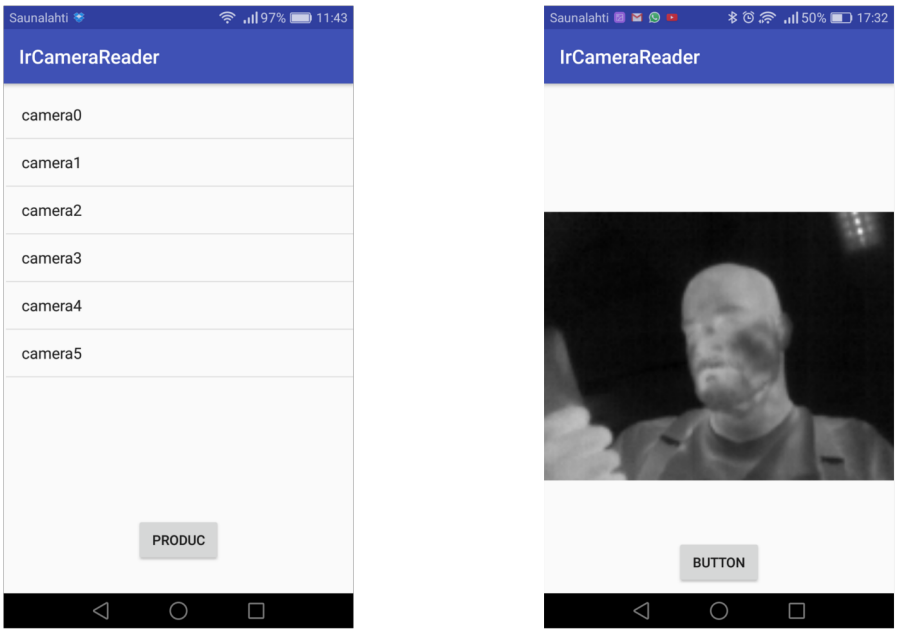


Figure 16: Android Client

The other activity makes the a Websocket connection and process the data in real time so the different frames can be visualized in gray-scale (figure 16).

To achieve this Android allows to extend UI elements with new behavior. In this case ”ImageView” class is extended to add a Websocket connection and fill the image with an array of binary data from this connection (same than in the python application section 3.3.1), in listing 13 the ”CameraView” class meets this behavior using ”WebsocketConnection” to get data from sensors (listing 15) and ”HighCamera” to represent a frame of the image (listing 14). All of these listing can be found in appendix 2.

3.4 Server Side Methods

On this section describes the different approaches tested to created the middle point between sensors and clients. The server side application must be hosted somewhere, in this case a cloud environment described in the next section.

3.4.1 The communication channel

Regardless what technology it is used for coding, build and maintain the cloud application, and before start coding it should define how different components are related between each other. Let's start by defining the following entities:

- Sensor: It represents a single sensor connected to a raspberry pi, although it could a normal computer. The sensor itself does not connect to networks, thus it needs a middle hardware but it is considered as whole "entity" here.
- Client: A client can be anything that can connect to a sensor by HTTP request. A desktop computer, a smart-phone, etc. The client must create frames from sensor and send commands using a Websocket.
- Cloud: It is what holds all the sensor entities and their clients on it. The channel should support an unlimited number of sensors which can hold an unlimited number of clients.

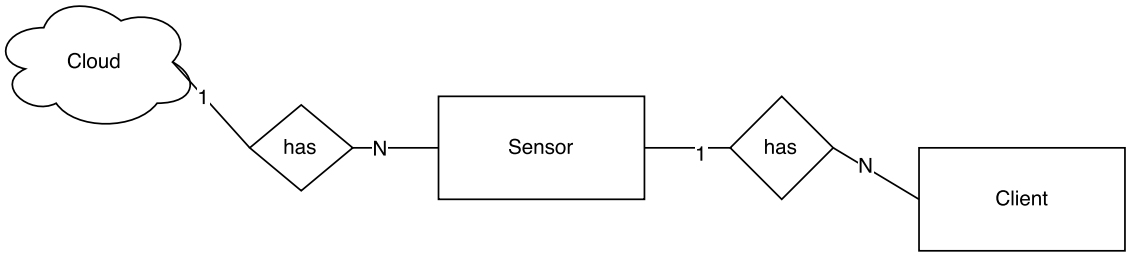


Figure 17: Entity Relationship Diagram of the communication channel

The channel should support an unlimited number of sensors which can hold an unlimited number of clients. The diagram in figure 17 represents the general view of how entities are related to each other. This helps to develop a UML Class Diagram which specifies how classes in an OOP language are related to each other. In figure 18 there is such diagram simplified. On the final implementation the server side application was written in Javascript which is the Language for Node.js applications.

3.4.2 Python Flask

Flask is a Python framework for web development which allows to write web applications back-ends and is specially useful to create web APIs. this application use an HTTP

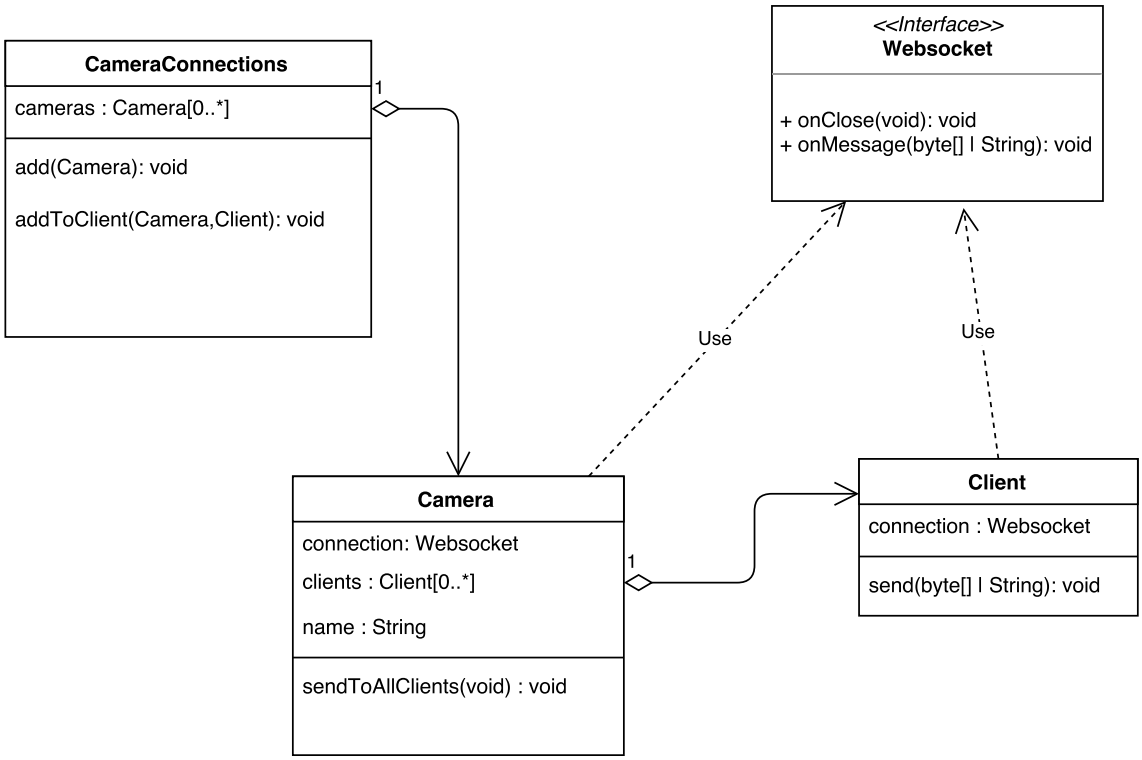


Figure 18: UML diagram

approach as described in section 2.3.2 to communicate client side and sensor side applications must implement the mentioned approach as well.

This keeps a queue buffer of image data and commands, so that both sensor and client have to request continuously even though when there is no new data. In listing 7 a simplification of this Flask application is shown, the buffer and commands are hold in the `data_queue` and `parameters`.
Using the `/video/buffer` endpoint it is possible to send data to it using a 'POST' HTTP requests continuously from the sensor side and to send 'GET' requests to get image data in the client side, if a client wants to send a command (called parameter in this application) it can be sent as query parameter in the 'GET' request, for example `www.example.com/video/buffer?command0=value0&command1=value1&...`

A good thing about flask is its straightforward API as seen in listing 7, defining a route to an endpoint is very easy because takes advantage of the decorators `@app.route` and result in a quite clear code.

```
app = Flask(__name__)
parameters = {}
data_queue = Queue(5)

@app.route('/video/buffer', methods=['POST'])
def submit_buff():
    data = request.data
    logging.debug("data:%s", data)
    if data_queue.full():
        logging.debug('queue is full. dropping 1')
        data_queue.get() # drop 1 buffer

    data_queue.put(data)
    return jsonify(**parameters)


@app.route('/video/buffer', methods=['GET'])
def obtain_buff():
    global parameters
    print request.args
    for k, v in request.args.items():
        parameters[k] = v

    if data_queue.empty():
        logging.debug('queue is empty, sending 0 ...')
        return '0'
    else:
        return data_queue.get()
```

Listing 7: HTTP - buffer approach

3.4.3 Node js Server Application

Node.js is a Javascript runtime which allows to create server-side applications, the main reason of why one would choose Node.js over other well know options such as Apache + PHP is the non-blocking model Node.js is based on. This let make asynchronous code easily and it is quite fast [19, p.12] [20]. In addiction to this there are other options to consider working with Node.js:

- It has big community.
- It has likely the biggest open source library on Internet, accessible through Node Packet Manager (NPM).
- most of cloud providers offer it out of the box.
- It is very easy to start up and configure unlike options.

To work with both Websocket and HTTP protocols in a Node.js application it is possible

to use these quite well know open source libraries:

- ws. It is claimed to be the fastest Websocket library https://www.npmjs.com/package/ws
- Express. Lightweight web framework for node. Probably the most used node web framework https://www.npmjs.com/package/express

Although this libraries are focused for web applications the clients described in section 3.3 do not need HTML characteristics . However in the future a HTML client application could be developed using the API of this Node.js application.

Since both protocols must be listen the same port, a HTTP server must be created using express then integrating it with the Websocket one and finally listening to the desired port, see listing 8.

```
1  const express = require('express');
2  const WebSocket = require('ws');
3  const http = require('http');
4
5  const port = process.env.PORT
6  const ip = '0.0.0.0';
7
8  /* http server */
9  const express = express();
10 const server = http.createServer(express);
11     // ... handle http requests ...
12
13 /* websocket server extends the http server */
14 var wss = new WebSocket.Server({
15     server: server,
16     // other websocket configuration ...
17 });
18
19  wss.on('connection', function connection(ws) {
20     // ... handle websocket requests ...
21 });
22
23 server.listen(port, ip);
```

Listing 8: html and websocket server

Express and ws have events to handle the connections using callbacks to for any behavior one wants to add.

A complete view of the main file application can be consulted in appendix 3 listing 16

3.5 Testing Methods

Testing is an important part of Software development process it gives insight about the quality and how much the software is error-prone, however creating test and maintaining after refactoring it is also a time consuming practice and since this project is carried out by one person rather than a team of developers there were added test only in the server-side of the whole communication system.

3.5.1 Unit Testing

Unit Testing is about testing modules or units which are single pieces of software that should work independently of others. In OOP this is usually a class, but it could be a single function or method. It is the developer who should define what is a single unit.

3.5.2 Acceptance Testing: Robot Framework

In order to test the server-side application with arbitrary number of clients and sensors with different configuration and backgrounds an acceptance testing approach is very convenient; acceptance testing features a more structured and complex way of testing which allows a efficient way of pinpointing application failure, automation and reusability [21].

For this purpose Robot Framework is a great tool, it is a open source for general purpose test automation. It is very flexible and it allows create test using a human friendly keywords and generate complete reports and logs about the test results. It can also be extended by adding existing libraries or creating them using Python or Java [22]. A simple test file can be seen in listing 9
Using the libraries already in the client python application in section 3.3.1 and the Robot Framework API I created a library to test Websocket connections. Those connection are held in an array, created and deleted for every single test, see listing 19 for the whole library code in the appendix 3.

The library provide keywords like the ones in the listing 9 but a group of test cases have to be defined in addiction to define some local keywords to reuse behavior. see listing 18

```
 1 Documentation     A test suite with a single test for valid login.
 2 Resource          resource.txt
 3
 4 *** Test Cases ***
 5 Valid Login
 6 Open Browser To Login Page
 7 Input Username    demo
 8 Input Password    mode
 9 Submit Credentials
10 Welcome Page Should Be Open
11 [Teardown]    Close Browser
```

Listing 9: A example of robot framework

in appendix 3 for all test cases defined to test the server-side application. On this file it can be noticed several sections between triple asterisks:

- Variables. To define global variables which syntax is "`${variable_name}`".
- Settings. Define libraries which in addiction to my own Websocket library there is also other to run processes, invoke operative system commands, HTTP requests, etc. Provide actions at beginning and the end of every test or even at the beginning and end of the whole test suite.
- keywords. Here it is defined custom keywords out of other keywords, they can accept arguments and return values.
- Test Cases. The last section show every test case. At the end will be shown either if the test was passed or failed.

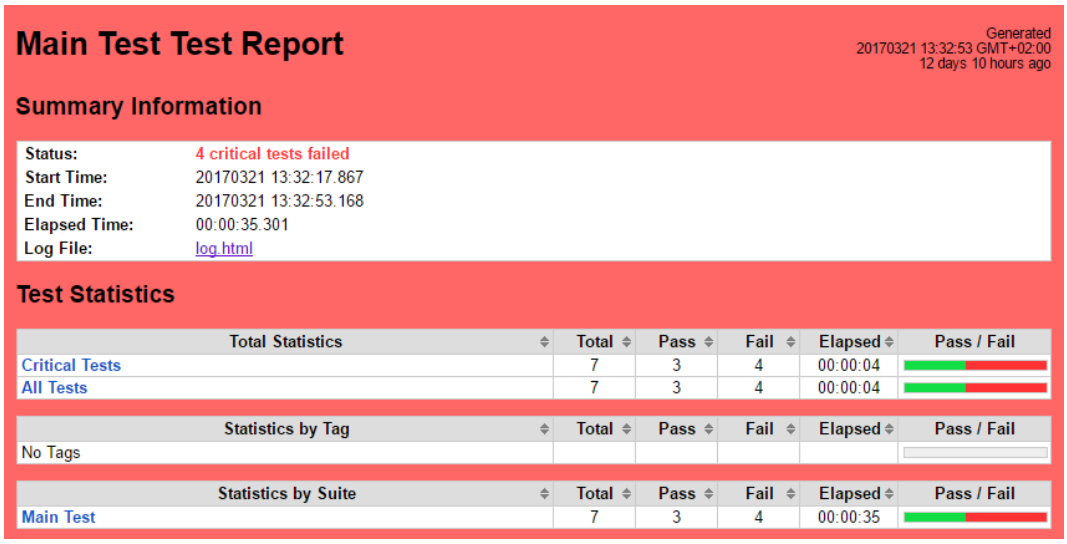
After the execution of tests an HTML report file is generated (figure 19) along to a log one (figure 20)

The Report is general view of what happened during the test execution.

The log file contain a more detailed information about the tests.

3.6 Cloud methods

While the Server Side methods section 3.4 describes the application itself, it does not tell anything about where and how to host the code so that it is available everywhere from Internet.



Main Test Test Report

Generated
20170321 13:32:53 GMT+02:00
12 days 10 hours ago

Summary Information

Status:	4 critical tests failed
Start Time:	20170321 13:32:17.867
End Time:	20170321 13:32:53.168
Elapsed Time:	00:00:35.301
Log File:	log.html

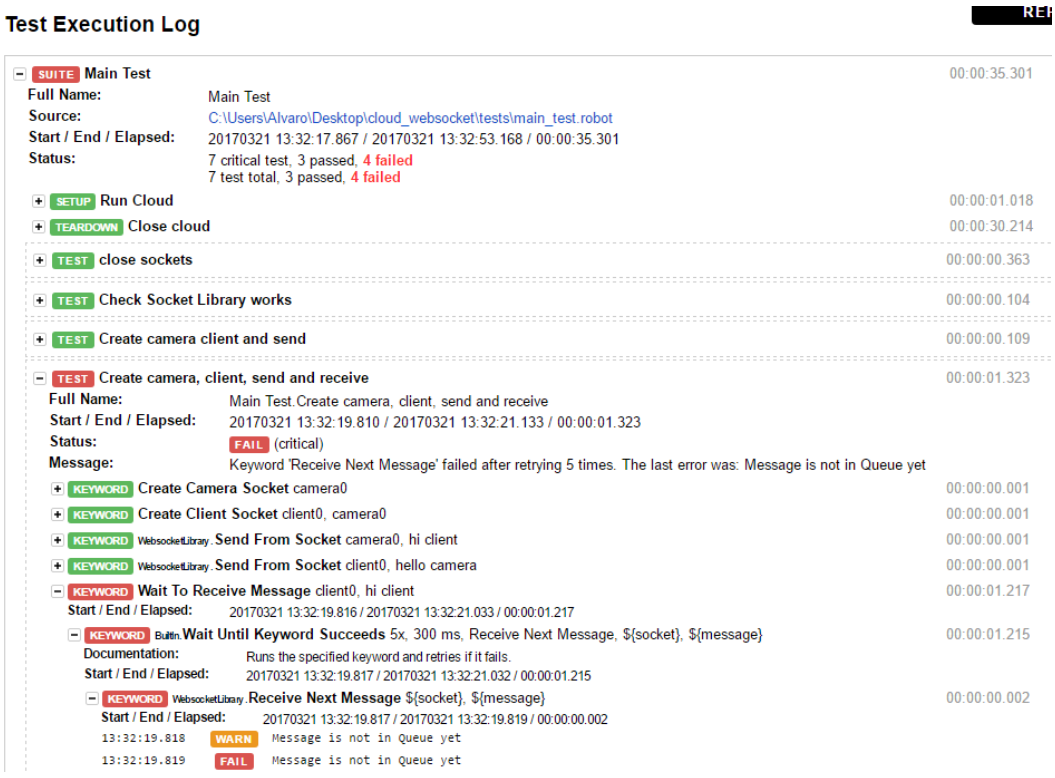
Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	7	3	4	00:00:04	
All Tests	7	3	4	00:00:04	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Main Test	7	3	4	00:00:35	

Figure 19: Test Report



Test Execution Log

SUITE Main Test 00:00:35.301
Full Name: Main Test
Source: C:\Users\Alvaro\Desktop\cloud_websocket\tests\main_test.robot
Start / End / Elapsed: 20170321 13:32:17.867 / 20170321 13:32:53.168 / 00:00:35.301
Status: 7 critical test, 3 passed, 4 failed
7 test total, 3 passed, 4 failed

SETUP Run Cloud 00:00:01.018
TEARDOWN Close cloud 00:00:30.214
TEST close sockets 00:00:00.363
TEST Check Socket Library works 00:00:00.104
TEST Create camera client and send 00:00:00.109
TEST Create camera, client, send and receive 00:00:01.323
Full Name: Main Test.Create camera, client, send and receive
Start / End / Elapsed: 20170321 13:32:19.810 / 20170321 13:32:21.133 / 00:00:01.323
Status: FAIL (critical)
Message: Keyword 'Receive Next Message' failed after retrying 5 times. The last error was: Message is not in Queue yet

KEYWORD Create Camera Socket camera0 00:00:00.001
KEYWORD Create Client Socket client0, camera0 00:00:00.001
KEYWORD WebsocketLibrary . Send From Socket camera0, hi client 00:00:00.001
KEYWORD WebsocketLibrary . Send From Socket client0, hello camera 00:00:00.001
KEYWORD Wait To Receive Message client0, hi client 00:00:01.217
Start / End / Elapsed: 20170321 13:32:19.816 / 20170321 13:32:21.033 / 00:00:01.217
KEYWORD BuiltIn . Wait Until Keyword Succeeds 5x, 300 ms, Receive Next Message, ${socket}, ${message} 00:00:01.215
Documentation: Runs the specified keyword and retries if it fails.
Start / End / Elapsed: 20170321 13:32:19.817 / 20170321 13:32:21.032 / 00:00:01.215
KEYWORD WebsocketLibrary . Receive Next Message ${socket}, ${message} 00:00:00.002
Start / End / Elapsed: 20170321 13:32:19.817 / 20170321 13:32:19.819 / 00:00:00.002
13:32:19.818 WARN Message is not in Queue yet
13:32:19.819 FAIL Message is not in Queue yet

Figure 20: Test Logs

”The cloud” is quite a broad concept which usually includes different service models as Software As A Service (SaaS) which are hosted applications accessible from client applications, Platform As A Service (PaaS) which provides services for developers to create applications such as SaaS ones, however the developer do not have access to the cloud infrastructure like the servers or the network, Infrastructure As A Service (IaaS). Similar to PaaS but with more control over the platform [23].

OpenShift and Docker

OpenShift is a platform for the deployment of web applications and services. It takes advantage of technologies such as Kubernetes and Docker to run and manage application in containers [24]. Nokia (which collaborates actively in this project) provides PaaS as part of their Nokia Innovation Platform which aims to create solutions for IoT [3].

The process of updating the server side application is through a OpenShift mechanism called Source-to-Image (S2I) which allow creating containers from the application source code without using Docker files, it its simplest form a developer provides a Git repository to the platform and OpenShift takes cares of everything: building, deploying, routing, etc [24, 5].

3.7 Other Tools used

3.7.1 Control Version

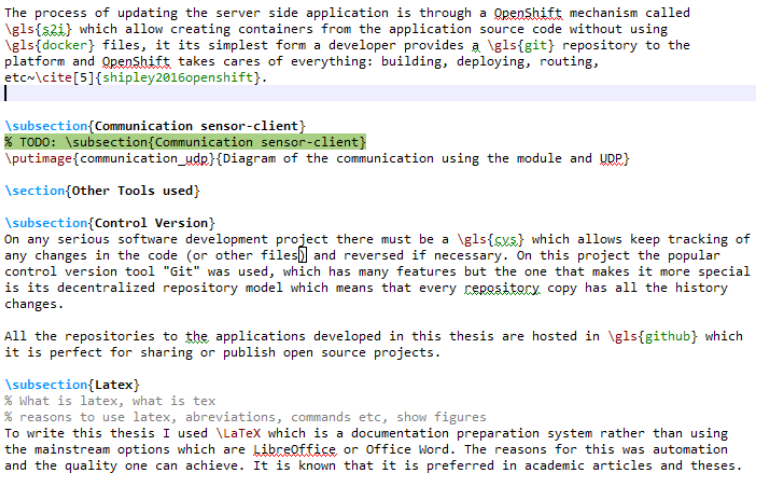
On any serious software development project there must be a Control Version System (CVS) which allows keep tracking of any changes in the code (or other files) and reversed if necessary. On this project the popular control version tool ”Git” was used, which has many features but the one that makes it more special is its decentralized repository model which means that every repository copy has all the history changes.

All the repositories to the applications developed in this thesis are hosted in GitHub which it is perfect for sharing or publish open source projects.

3.7.2 Latex

To write this thesis LaTeXwas used which is a documentation preparation system rather than using the mainstream options which are LibreOffice or Office Word. The reasons for this was automation and the quality that can achieve.

In figure 21 it can be observed how the code looks like, it is just raw text with no style at



```
The process of updating the server side application is through a OpenShift mechanism called
\gls{s2i} which allow creating containers from the application source code without using
\gls{docker} files, it its simplest form a developer provides a \gls{git} repository to the
platform and OpenShift takes cares of everything: building, deploying, routing,
etc~\cite[5]{shipley2016openshift}.


\subsection{Communication sensor-client}
% TODO: \subsection{Communication sensor-client}
\putimage{communication_udp}{Diagram of the communication using the module and UDP}

\section{Other Tools used}

\subsection{Control Version}
On any serious software development project there must be a \gls{cvs} which allows keep tracking of
any changes in the code (or other files) and reversed if necessary. On this project the popular
control version tool "Git" was used, which has many features but the one that makes it more special
is its decentralized repository model which means that every repository copy has all the history
changes.

All the repositories to the applications developed in this thesis are hosted in \gls{github} which
it is perfect for sharing or publish open source projects.

\subsection{Latex}
% What is latex, what is tex
% reasons to use latex, abreviations, commands etc, show figures
To write this thesis I used \LaTeX which is a documentation preparation system rather than using
the mainstream options which are LibreOffice or Office Word. The reasons for this was automation
and the quality one can achieve. It is known that it is preferred in academic articles and theses.
```

Figure 21: How the document looks like

all and it has to be compiled to generate a Portable Document Format (PDF) document.

4 Results

The results of this thesis is a set of pieces of software which together form the communication channel, being its central part the server-side hosted in cloud accessible through the web API described on section 4.2.

4.1 Software

After solving some bugs in the server-side and the raspberry pi part the communication system can work continuously 24 hours a day with several cameras as 5.4.2017 the longest test in a sensor in the system has been 4 days and 19 hours. In figure 22 can be seen a client application with sensors on different locations.

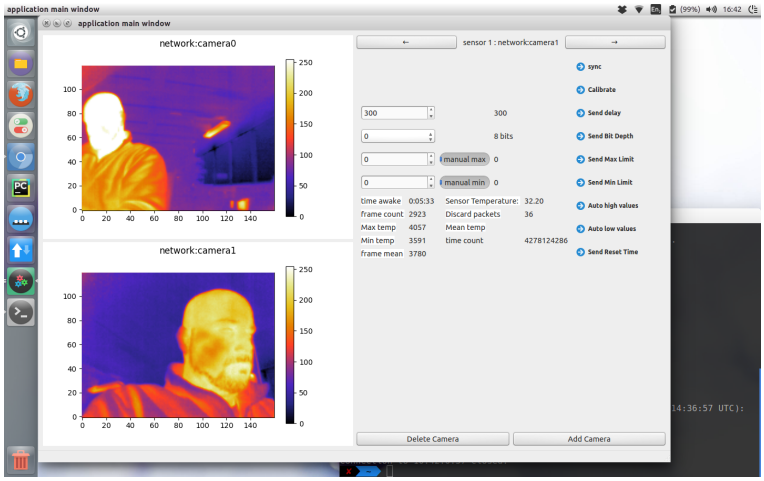


Figure 22: Multiple sensors connected to a single client

As stated in chapter 3 all the code is in GitHub repositories, here is a list with every repository for most of the software developed in this project.

- Android client application: https://github.com/alvaro893/android-ir-sensor-client
- Server-side Node.js application: https://github.com/alvaro893/cloud_websocket
- Python client application: https://github.com/alvaro893/sensor_reader
- Sensor Reader application on raspberry pi side: https://github.com/alvaro893/sensor-reader/tree/raspberry

- This Thesis: https://github.com/alvaro893/bachelors-thesis

The code shown in the appendixes is in these repositories and up-to-date.

4.2 Web Api

In order to provide communication with both sensors and clients a web API can be the most convenient way to expose the services of the application providing certain endpoints to register sensors, clients and to access information about the application itself, the information retrieved is in JSON format.

The endpoints are defined as Uniform Resource Identifier (URI)s. A complete URI looks like this:

```
<protocol>://<domain>:<protocol><path>?<parameters>
```

Domain is provided by the cloud service, if the server-side application is running locally in a personal computer domain is ”localhost”. Paths accepted by the API are shown in table 3

path	protocol	description
`/client`	Websocket	Register a client
`/camera`	Websocket	Register a sensor
`/cams`	HTTP GET	receive a JSON of information

Table 3: Endpoints of the web API

`/client` and `/camera` parameters also accept query parameters shown in table 4.

parameter	type	description
`pass`	alphanumeric	password to access
`camera_name`	alphanumeric	camera to connect or to register

Table 4: parameters of the web API

For example `ws://localhost:8080/camera?pass=d8n2d0&camera_name=corner-camera`

4.3 Discarded Methods

4.3.1 android smart-phone gateway solution

This method was purely to test the sensors and to seek for a suitable network protocol rather than as a serious solution for the purposes of this thesis, nevertheless the outcomes were satisfactory and opened the door to a deeper understanding on how USB communication can work using a High level language as Java.

4.3.2 LTE module solution

This method could have been the ideal solution to read a sensor and connect it to Internet, but there were multiple issues regarding this method:

- It turned out that the module did not support the LTE frequency bands of the mobile provider Netleap see section 2.2 of chapter 2.
- It was hard to set-up a developing environment for it.

4.3.3 UDP protocols

There were many test to use UDP for communications but for the issues mentioned in section 2.3.1 of chapter 2 and because Websocket worked very good. any UDP was dropped.

4.3.4 python flask server-side application

At the end the Node.js application turned out to work better and faster. so the flask application was not further developed.

5 Discussion

On this chapter is discussed how the results may be used, specially alongside Computer Vision. The sections here are more based on speculation rather than experimentation and proposed lines of research.

5.1 Use in the Nokia Innovation Platform

The Nokia Innovation Platform is a trial environment for teams and star-ups that are related to IoT projects specially projects that use cell-phone network as LTE. This project is part of the Nokia Innovation Platform hence there is some ongoing use cases. Some of the sensors has been placed on different places to test its functionality and investigate further the uses of this communication channel.

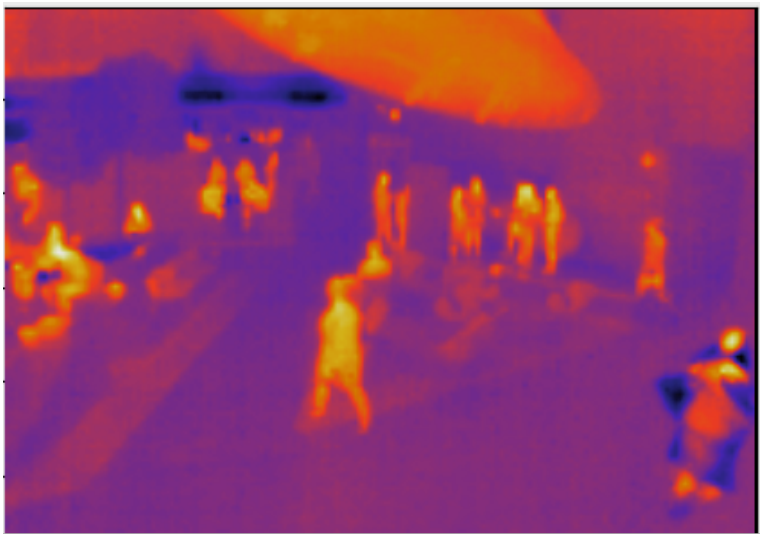


Figure 23: People in a room

On the figure 23 some people can be seen clearly and in the figure 24 is shown the same situation plus running a Computer Vision software to detect how many people in the room.

Computer vision can be used to track people and research their behavior storing big amounts of tracking data may allow to make predictions of many sorts. On the next sections some scenarios are proposed where this could be useful.

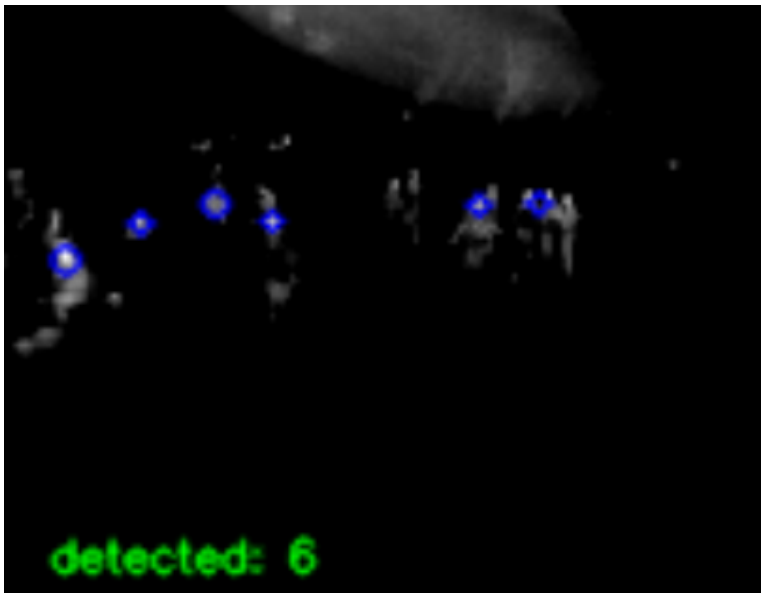


Figure 24: Analyzed image to detect heat points in the room

On crowded surfaces such as shopping centers, stadiums, summer festivals may be useful to observe patterns and even detect or predict dangerous situations. This is an interesting research topic and could be a continuation of this final year project.

5.2 Use in health care

In this section different uses in health care of the sensor communication channel will be discussed. As mentioned on the introduction chapter 1 these sensors do not capture light but heat, thus making patients impossible to recognize from a distance, this can be considered an advantage since most of the people do not feel comfortable with the idea of being watched or video-recorded. Also when using Computer Vision techniques an IR image can be easier for a computer to ”understand” in other words to analyze especially when it is possible to add limits to the maximum and minimum temperature so that a range of temperature can be interesting while the rest is removed. figure 25.

On table 2 in chapter 3 the available commands to the sensor are shown, we can set maximum and minimum temperatures or make it back to automatic.

In addiction to set the minimums and maximums it is possible to change the bit density of the image so we get less data and probably making the image analysis a little easier as it is shown in figure 26.

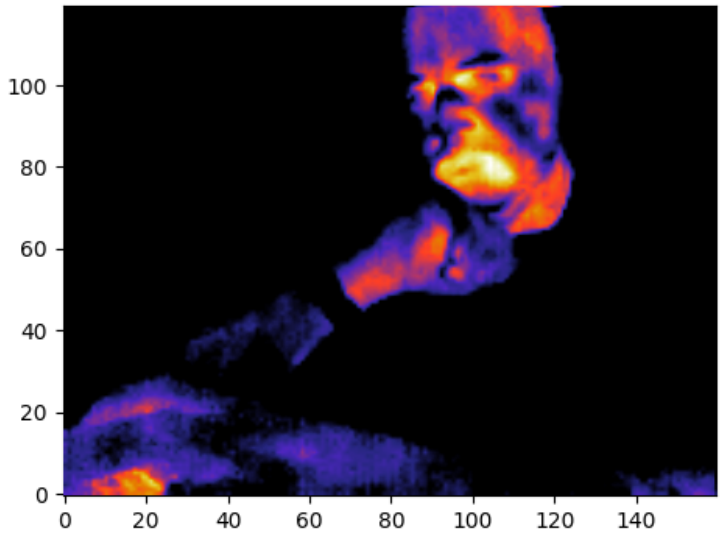


Figure 25: Minimum set to human temperature

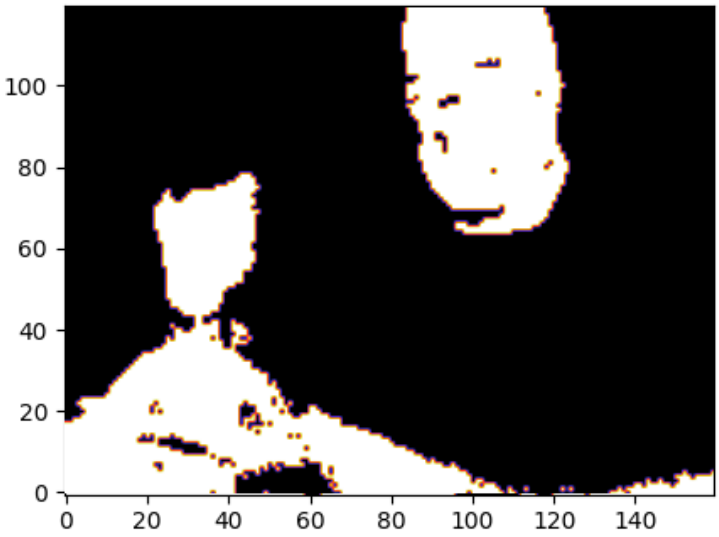


Figure 26: 2bit version of the image

Another approach could be use ir image to see things that usually are not visible at a glance like for example veins and arteries as seen in figure 27.

Unfortunately on this thesis it were not enough time to do a proper research of health care uses. Nevertheless on the next sections there is some propositions to for further investigation.

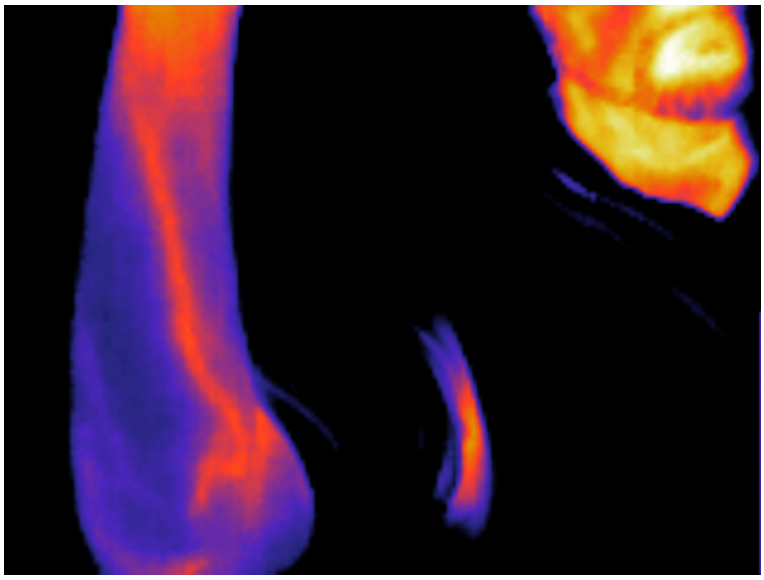


Figure 27: arteries and veins in human arm

5.2.1 Bed patient care

A use case may be for monitoring patients on bed, either hospital or home bed. Using image analysis software nurses or doctor could obtain data from multiple patients at the same time. since the sensors can detect heat this could trigger some alarm on high temperature or if the patient is missing for a long period of time.

Also as discussed in previous sections it may be interesting to seek patterns how a patients moves through time and even how long patients are not in bed.

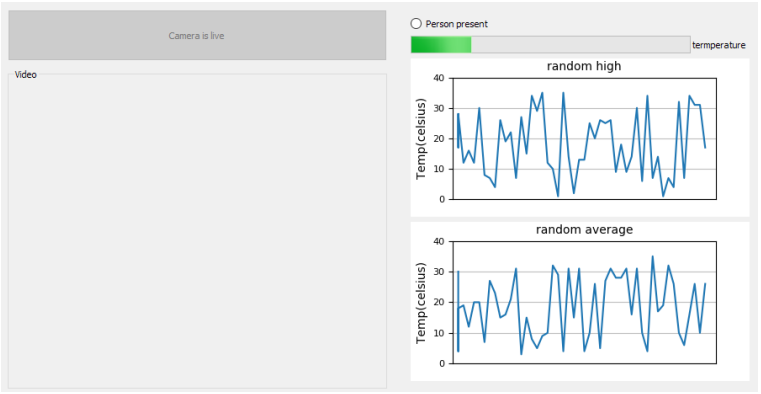


Figure 28: Analysis program prototype

In the figure 28 we can see a prototype with temperature graphics and other indicators.

5.2.2 Nursing homes / Psychiatric hospitals

The same principle discussed in section 5.1 may be applied to nursing or residential homes where there is a number of people inside a room or outside in a backyard. Then trough Computer Vision it can be detected how many persons there are, if someone is too still or passed out, etc. This detection could trigger certain alarms which will warn the caretakers in the facility

As in nursing homes, psychiatric hospitals could have this kind of system. It could detect potential dangerous situations in this case.

5.3 Other uses

5.3.1 Self-driven vehicles

Self-driven public transport vehicles could benefit with this system too, since that there is no driver nor any other worker on the vehicle, accessible remotely IR sensor can give valuable data to an hypothetically station in charge of the security of such vehicles and trigger an alarm in case of any dangerous situation.

Figure 29: self-driven bus in Otaniemi

On figure 29 is the ”Robot bus” moving around Otaniemi campus in Espoo.

5.3.2 General surveillance

Surveillance is usually help by regular light based cameras plus sometimes some IR LEDs which allows them see during night time, However since humans emit heat they can be easily spotted with IR sensors instead.

5.4 Possible additions in the future

5.4.1 3D Heat map using multiple sensors

When using several sensors, it could be possible to create 3D heat map with all the data similar to figure 30.

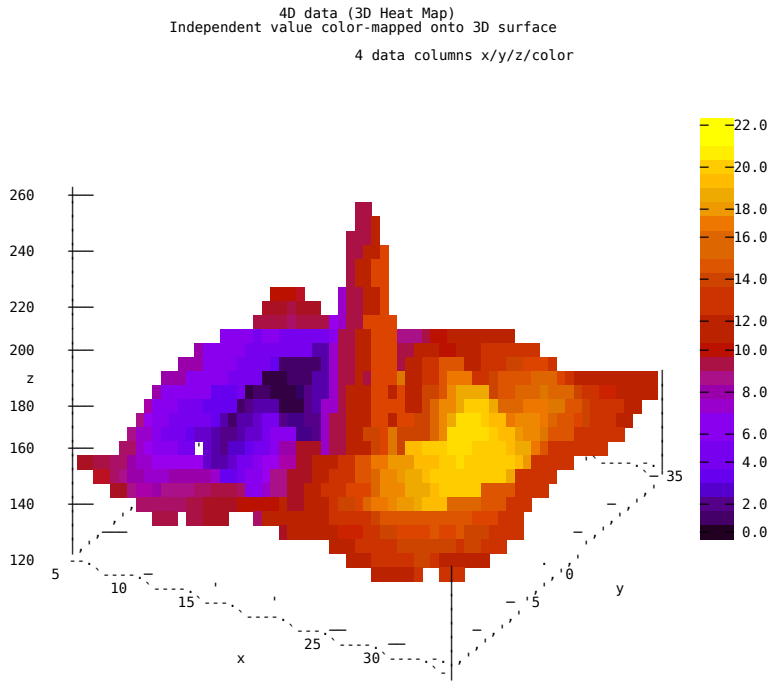


Figure 30: Heat map concept

This can be used to have a better understanding of how heat behaves in a particular area.

5.4.2 HTML5 application

Since this project uses already web technologies such as Websocket it will not difficult to create an HTML5 client application for connecting to the different sensors able to do the

same as the python client, however the user does not need to install anything to make it work, only an up to day Internet browser.

5.4.3 Using RTP to broadcast video to clients

The server-side application could be extended to support UDP protocols as RTP which can be used to multi-cast video stream to a big number of clients that for example need the video image but no control the sensor [25, 298]. Commands can be still sent through Websocket

6 Conclusions

As stated in the business challenge this project was about ”create a effective communication channel between ir sensors and clients such as computers, laptops, etc and use it on Well-being services” which can be concluded that it is done even beyond the original requirements. Chapters 4 and 5 have the information on what was done and propositions to use the ir cameras with the communication system. It was intended to deepen much more on the use cases, unfortunately there was no time for more research on this. Perhaps other students' thesis might continue this part.

Note that although the project is done, development may continue as an Nokia Innovation Platform project with LeViteZer.

Please refer to section 4.1 in chapter 4 to obtain the last version of the code created in this project.

References

1 Eurostat. Increase in the share of the population aged 65 years or over between 2005 and 2015;. Available from: http://ec.europa.eu/eurostat/statistics-explained/index.php/File:Increase_in_the_share_of_the_population_aged_65_years_or_over_between_2005_and_2015_(percentage_points)_YB16.png.

2 levitezer; 2017. Available from: http://www.levitezer.com.

3 Nokia. Nokia Innovation Platform; 2017. Available from: https://networks.nokia.com/innovation/platform.

4 Hunt C. TCP/IP Network Administration: Help for Unix System Administrators. O'Reilly Media; 2002. Available from: https://books.google.fi/books?id=A_LL2LQASdoC.

5 Timo Knuutila TK. Net Leap Network. In: Net Leap Network. http://digi.aalto.fi/en/midcom-serveattachmentguid-1e44979621ab084497911e49b7eb59a384bb1cab1ca/netleapnetworkpresentation-digibreakfast.pdf: Aalto University; 2014. p. 10.

6 Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, et al.. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1; 1999. Available from: https://www.ietf.org/rfc/rfc2616.txt.

7 Melnikov A, Fette I. The WebSocket Protocol; 2011. RFC 6455. Available from: https://tools.ietf.org/html/rfc6455.

8 Google. Android Documentation - Services;. Available from: https://developer.android.com/guide/components/services.html.

9 mik3y (https://github com/mik3y). usb-serial-for-android;. Available from: https://github.com/mik3y/usb-serial-for-android#usb-serial-for-android.

10 Oracle. JAVA PLATFORM, MICRO EDITION (JAVA ME); 2017. Available from: http://www.oracle.com/technetwork/java/embedded/javame/index.html.

11 Raspberry pi Foundation. RASPBERRY PI 3 MODEL B;. Available from: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

12 Python Software Foundation. Python Documentation - multiprocessing;. Available from: https://docs.python.org/2/library/multiprocessing.html.

13 Rodola G. psutil documentation; 2017. Available from: https://pythonhosted.org/psutil/.

14 Python Software Foundation. Subprocess documentation; 2017. Available from: https://docs.python.org/2/library/subprocess.html.

15 Notes E. What is LabVIEW?; 2016. Available from: https://www.electronics-notes.com/articles/test-methods/labview/what-is-labview.php.

16 Riverbank Computing Limited. What is PyQt?; 2016. Available from: https://riverbankcomputing.com/software/pyqt/intro.

17 John Hunter EFMD Darren Dale, the Matplotlib development team. backend qt4agg; 2016. Available from: http://matplotlib.org/api/backend_qt4agg_api.html.

18 John Hunter EFMD Darren Dale, the Matplotlib development team. Interpolation methods; 2016. Available from: http://matplotlib.org/examples/images_contours_and_fields/interpolation_methods.html.

19 Dayley B. Node.js, MongoDB, and AngularJS Web Development. Developer's Library. Pearson Education; 2014. Available from: https://books.google.fi/books?id=8kTCAwAAQBAJ.

20 Node js Foundation. Overview of Blocking vs Non-Blocking; 2017. Available from: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/.

21 Bisht S. Robot Framework Test Automation. Packt Publishing; 2013. Available from: https://books.google.fi/books?id=_RO8AQAAQBAJ.

22 Robot Framework Foundation. Robot Framework introduction; 2017. Available from: http://robotframework.org/#introduction.

23 Mell P, Grance T. The NIST Definition of Cloud Computing. National Institute of Standards and Technology: U.S. Department of Commerce; 2011. 800-145.

24 Shipley G, Dumpleton G. OpenShift for Developers: A Guide for Impatient Beginners. O'Reilly Media; 2016. Available from: https://books.google.fi/books?id=qKvODAAAQBAJ.

25 Hardy D, Malleus G, Mereur JN. Networks: Internet · Telephony · Multimedia. Springer Berlin Heidelberg; 2013. Available from: https://books.google.fi/books?id=GZPzCAAAQBAJ.

Acknowledgments

I want to thank to Jukka Honkaniemi and Tero Nurminen (Metropolia) for let me be part of this project and their guidance. Also I want to Thanks to Kim Janson (LeViteZer) for all I learn during last months in this project.

1 Sensor reader listings

```
1  import thread
2  import logging
3  import psutil
4  from multiprocessing import Process
5  from serial import Serial, SerialException
6  from Constants import VERY_HIGH_PRIORITY, HIGH_PRIORITY
7
8  class Serial_reader(Serial):
9  """ This class read data from sensor in a Thread """
10     def __init__(self, pipe, port):
11         Serial.__init__(self, port=port, baudrate=115200)
12         self.pipe = pipe
13         self._start_process()
14
15     def _start_process(self):
16         process = Process(name="SerialProcess", target=self._run, args=())
17         process.daemon = True
18         process.start()
19         try:
20             psutil.Process(process.pid).nice(VERY_HIGH_PRIORITY)
21         except psutil.AccessDenied as e:
22             psutil.Process(process.pid).nice(HIGH_PRIORITY)
23
24     def _get_data(self):
25         one_byte = self.read(1)
26         n_bytes = self.in_waiting
27         return one_byte + self.read(n_bytes)
28
29     def _send_data(self):
30         while self.is_open:
31             print "waiting for commands"
32             data = self.pipe.recv()
33             print data
34             self.write(data)
35
36     def _run(self):
37         thread.start_new_thread(self._send_data, ())
38         while self.is_open:
39             try:
40                 data = self._get_data()
41                 self.pipe.send(data)
42             except SerialException as e:
43                 logging.error(e.message)
```

```
                self.stop()
                break

    def stop(self):
        if self.is_open:
        self.close()
```

Listing 10: Serial Reader Class

```
import logging
import thread
from websocket import WebSocketApp, ABNF
from Constants import URL, CAMERA_PATH, PARAMETERS

class WebSocketConnection(WebSocketApp):
    def __init__(self, pipe, url=URL + CAMERA_PATH + PARAMETERS):
    WebSocketApp.__init__(self, url,
    on_message=self.on_message,
    on_error=self.on_error,
    on_close=self.on_close,
    on_open=self.on_open)
    self.open_connection = False
    self.pipe = pipe

    def on_message(self, ws, message):
        logging.warn("received command:%s, %d bytes", message[0], len(message))
        self.pipe.send(message)

    def on_error(self, ws, error):
        logging.error(error)

    def on_close(self, ws):
        self.open_connection = False
        logging.warn("### closed ###")

    def on_open(self, ws):
        self.open_connection = True
        logging.warn("opened new socket")

        def run():
            while (self.open_connection == True):
                data = self.pipe.recv()
                self.send_data(data)

        thread.start_new_thread(run, ())

    def stop(self):
        self.open_connection = False
```

```
40
41      def send_data(self, data):
42          if self.open_connection and len(data) != 0:
43          self.send(data, opcode=ABNF.OPCODE_BINARY)
44
45      def set_pipe(self, pipe):
46          self.pipe = pipe
```

Listing 11: Websocket Class

```
1  from thread import start_new_thread
2  from subprocess32 import call
3  import logging
4
5  def shutdown():
6      run_command_async("/sbin/poweroff")
7
8  def reboot():
9      run_command_async("/sbin/reboot")
10
11 def update():
12     run_command_async("./build.sh")
13
14 def test():
15     run_command_async("sleep", "3")
16
17 commands = {
18 'rs': shutdown,
19 'rr': reboot,
20 'ru': update
21 }
22
23 def run_command_async(*args):
24     def run(*args):
25         result = call(args)
26
27     start_new_thread(run, args)
28
29
30 def is_raspberry_command(s):
31 """ Check this is a command for the raspberry rather than the sensor. First letter
       must be an 'r'"""
32     if s[0] == 'r':
33         command = commands.get(s)
34         if callable(command):
35             command()
36         else:
37             logging.warn("command not found: %s" % s)
```

```
38              return True
39      else:
40          return False
```

Listing 12: Subprocess example

2 Android Client listings

```
1  package es.alvaroweb.ircamerareader.wscameraview;
2
3  import android.content.Context;
4  import android.graphics.Bitmap;
5  import android.graphics.Canvas;
6  import android.graphics.Color;
7  import android.support.annotation.Nullable;
8  import android.util.AttributeSet;
9  import android.util.Log;
10 import android.view.View;
11 import android.support.v7.widget.AppCompatImageView;
12 import java.util.Random;
13 import okio.ByteString;
14
15 /**
16 * Copyright (C) 2016 Alvaro Bolanos Rodriguez
17 */
18
19 public class CameraView extends AppCompatImageView implements WebsocketConnection.
       OnReceiveRow, HighCamera.FrameCallback, View.OnClickListener {
20
21     private static final String DEBUG_TAG = CameraView.class.getSimpleName();
22     private static Random random = new Random();
23     private Bitmap bitmap;
24     private int sizex = 160;
25     private int sizey = 120;
26     private WebsocketConnection websocketConnection;
27     private HighCamera highCamera;
28     private Runnable t;
29     private boolean reversed = true;
30
31
32     public CameraView(Context context) {
33         super(context);
34         initBitmap();
35     }
36
37     public CameraView(Context context, @Nullable AttributeSet attrs) {
38         super(context, attrs);
39         initBitmap();
40     }
41
42     private void initBitmap() {
```

```
43          bitmap = Bitmap.createBitmap(sizex, sizey, Bitmap.Config.ARGB_8888);
44          highCamera = new HighCamera(this);
45          this.setOnClickListener(this);
46      }
47
48      @Override
49      protected void onDraw(Canvas canvas) {
50          super.onDraw(canvas);
51          // Bitmap has to be set here. and this callback is called from UI
52          this.setImageBitmap(bitmap);
53      }
54
55
56      public void setImage(byte[][] array) {
57          boolean dimensionMaches = array[0].length == bitmap.getWidth() &&
58          array.length == bitmap.getHeight();
59
60          if (!dimensionMaches) {
61              Log.d("image", "doesn't match the dimension");
62              return;
63          }
64
65          for (int i = 0; i < bitmap.getHeight(); i++) {
66              for (int j = 0; j < bitmap.getWidth(); j++) {
67                  int pixel = convertByteToInt(array[i][j]);
68                  if (reversed) {
69                      bitmap.setPixel(j, bitmap.getHeight() - i - 1, Color.rgb(pixel
                            , pixel, pixel));
70                  } else {
71                      bitmap.setPixel(j, i, Color.rgb(pixel, pixel, pixel));
72                  }
73              }
74          }
75      }
76
77      public void setRandomImage(View view) {
78          byte[][] arr = new byte[sizey][sizex];
79          for (int i = 0; i < sizey; i++) {
80              for (int j = 0; j < sizex; j++) {
81                  arr[i][j] = ((byte) randint(Byte.MIN_VALUE, Byte.MAX_VALUE));
82              }
83          }
84          setImage(arr);
85      }
86
87      public void cleanImage() {
88          for (int i = 0; i < bitmap.getWidth(); i++) {
89              for (int j = 0; j < bitmap.getHeight(); j++) {
```

```
90                  bitmap.setPixel(i, j, Color.rgb(255, 255, 255));
91              }
92          }
93      }
94
95      private int convertByteToInt(byte b) {
96          return b & 0xff;
97      }
98
99      private int randint(int min, int max) {
100         return random.nextInt(max + 1 - min) + min;
101     }
102
103     public void connectTo(String uri) {
104         Log.d(CameraView.class.getSimpleName(), "uri received: " + uri);
105         websocketConnection = new WebsocketConnection(uri, this);
106     }
107
108     public void stopWebsocket() {
109         websocketConnection.close();
110     }
111
112     @Override
113     public void receiveRows(ByteString data) {
114         if (data.size() < 1) {
115             return;
116         }
117         highCamera.consumeData(data);
118     }
119
120     @Override
121     public void frameReady(byte[][] frame) {
122         this.setImage(frame);
123     }
124
125
126     @Override
127     public void onClick(View view) {
128         reversed = !reversed;
129     }
130
131     public interface UpdateArray {
132         void updateArray();
133     }
134 }
```

Listing 13: CameraView class, extending ImageView

```java
package es.alvaroweb.ircamerareader.wscameraview;

import android.util.Log;
import com.google.common.primitives.Bytes;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import okio.ByteString;

/**
* 00 01 02 04 ------
Data is arranged on 240(0xF0) rows of 84 bytes(FF FF FF n_row and 80 of data):
FF FF FF 01 <DATA>
FF FF FF 02 <DATA>
FF FF FF 03 <DATA>
...
FF FF FF F0 <DATA>
FF FF FF <TELEMETRY> (38 Bytes)
Where the 4th byte is the number of the row
Every row of the actual picture has 2 rows of the raw data
so the image is 160 x 120 (20198 Bytes)
*/

public class HighCamera {
    private static final int TELEMETRY_ROW_NUMBER = 240;
    private static final int BYTES_IN_ROW_NUMBER = 81;
    private static final String DEBUG_TAG = HighCamera.class.getSimpleName();

    private byte[][] frame;
    FrameCallback frameCallback;
    private byte[] remains;
    private byte[] delimiter = new byte[]{-1,-1,-1};

    public HighCamera(FrameCallback callback) {
        frame = new byte[120][160];
        frameCallback = callback;
        remains = new byte[]{};
    }

    public void consumeData(ByteString data){

        List<byte[]> pieces = delimiterData(Bytes.concat(remains, data.toByteArray
            ()), delimiter);

        int lastIndex = pieces.size() - 1;
        for(int i = 0; i < pieces.size(); i++){
            if(i == lastIndex) continue;
            processRow(pieces.get(i));
```

```
48          }
49          remains = pieces.get(lastIndex);
50      }
51
52
53      public void processRow(byte[] row){
54          if(row.length < BYTES_IN_ROW_NUMBER){
55              return;
56          }
57          int rowNumber = byteToInt(row[0]);
58          Log.d(DEBUG_TAG, "rownumber:"+rowNumber);
59          if (rowNumber < TELEMETRY_ROW_NUMBER){
60              getFrameData(rowNumber, row);
61          }else{
62              getTelemetryData(row);
63          }
64      }
65
66      private void getTelemetryData(byte[] row) {
67          // todo telemetry
68          frameCallback.frameReady(frame);
69      }
70
71      private void getFrameData(int rowNumber, byte[] row){
72          for(int i = 0; i < BYTES_IN_ROW_NUMBER - 1; i++){
73              int ind = i + 1;
74              int frameRow = rowNumber / 2;
75              int frameCol = rowNumber % 2 * (BYTES_IN_ROW_NUMBER -1)+ ind;
76              try{
77                  frame[frameRow][frameCol] = row[ind];
78              }catch (ArrayIndexOutOfBoundsException e){
79                  Log.e(DEBUG_TAG, e.getLocalizedMessage());
80              }
81          }
82      }
83
84      private int byteToInt(byte b){
85          return b & 0xff;
86      }
87
88
89      private List<byte[]> delimiterData(byte[] array, byte[] delimiter) {
90          List<byte[]> byteArrays = new LinkedList<>();
91          if (delimiter.length == 0) {
92              return byteArrays;
93          }
94          int begin = 0;
95
```

```
96          outer:
97          for (int i = 0; i < array.length - delimiter.length + 1; i++) {
98              for (int j = 0; j < delimiter.length; j++) {
99                  if (array[i + j] != delimiter[j]) {
100                     continue outer;
101                 }
102             }
103             byteArrays.add(Arrays.copyOfRange(array, begin, i));
104             begin = i + delimiter.length;
105         }
106         byteArrays.add(Arrays.copyOfRange(array, begin, array.length));
107         return byteArrays;
108     }
109
110     interface FrameCallback{
111         void frameReady(byte[][] frame);
112     }
113 }
```

Listing 14: HighCamera class, represents a frame

```
1  package es.alvaroweb.ircamerareader.wscameraview;
2
3  import android.util.Log;
4  import okhttp3.OkHttpClient;
5  import okhttp3.Request;
6  import okhttp3.Response;
7  import okhttp3.WebSocket;
8  import okhttp3.WebSocketListener;
9  import okio.ByteString;
10
11
12 /**
13 * Copyright (C) 2016 Alvaro Bolanos Rodriguez
14 */
15
16 public class WebsocketConnection extends WebSocketListener {
17     private static final String DEBUG_TAG = WebsocketConnection.class.
           getSimpleName();
18     private final WebSocket webSocket;
19     private final Request requestToServer;
20     private OnReceiveRow callback;
21     public WebsocketConnection(String serverURI, OnReceiveRow callback) {
22         OkHttpClient client = new OkHttpClient();
23         requestToServer = new Request.Builder().url(serverURI).build();
24         webSocket = client.newWebSocket(requestToServer, this);
25         this.callback = callback;
26     }
```

```
27
28      @Override
29      public void onOpen(WebSocket webSocket, Response response) {
30          super.onOpen(webSocket, response);
31          log("onOpen: " + response.message());
32      }
33
34      @Override
35      public void onMessage(WebSocket webSocket, String text) {
36          super.onMessage(webSocket, text);
37          log("onMessage: " + text);
38
39      }
40
41      @Override
42      public void onMessage(WebSocket webSocket, ByteString bytes) {
43          super.onMessage(webSocket, bytes);
44          log("onMessage: " + bytes.size() + "bytes received");
45          callback.receiveRows(bytes);
46      }
47
48      @Override
49      public void onClosing(WebSocket webSocket, int code, String reason) {
50          super.onClosing(webSocket, code, reason);
51          log("onClosing: " + reason + ", code:" + code);
52      }
53
54      @Override
55      public void onClosed(WebSocket webSocket, int code, String reason) {
56          super.onClosed(webSocket, code, reason);
57          log("onClosed: " + reason + ", code:" + code);
58      }
59
60      @Override
61      public void onFailure(WebSocket webSocket, Throwable t, Response response) {
62          super.onFailure(webSocket, t, response);
63          Log.e(DEBUG_TAG, "onFailure: " + t.getMessage());
64          t.printStackTrace();
65
66      }
67
68      public void send(byte[] bytes){
69          ByteString byteString = ByteString.of(bytes);
70          webSocket.send(byteString);
71      }
72
73      public void close(){
74          webSocket.close(1000, "fulfilled");
```

```
75      }
76
77      private void log(String s){
78          Log.d(DEBUG_TAG, s);
79      }
80
81      interface OnReceiveRow{
82          void receiveRows(ByteString bytes);
83      }
84  }
```

Listing 15: WebsocketConnection class, makes a websocket connection

3 Node.js application listings

```
1  "use strict";
2
3  const express = require('express');
4  var WebsocketConnections = require('./websocketConnections');
5  var WebSocket = require('ws');
6  var url = require('url');
7  var http = require('http');
8  var params;
9
10 console.log("version 1.0");
11 var port = process.env.PORT || process.env.port || process.env.
       OPENSHIFT_NODEJS_PORT || 8080;
12 var ip = process.env.OPENSHIFT_NODEJS_IP || process.argv[2] || '0.0.0.0';
13
14 var PASSWORD = process.env.WS_PASSWORD;
15 var camDataPath = "/camera";
16 var clientDataPath = "/client";
17 var camConnections = new WebsocketConnections.CameraConnections();
18
19 /** http server: base */
20 const app = express();
21 app.get('/cams', function(req, res){
22     res.send({ cams: camConnections.getInfo(), count: camConnections.count()});
23 });
24 const server = http.createServer(app);
25 main(server)
26
27 /** @function
28 *  @param {http.Server} server */
29 function main(server) {
30     /** websocket server extends the http server */
31     var wss = new WebSocket.Server({
32         verifyClient: verifyClient,
33         server: server
34     });
35
36     console.log("running on %s:%d", ip, port);
37
38     wss.on('connection', function connection(ws) {
39         var parsedUrl = url.parse(ws.upgradeReq.url);
40         var path = parsedUrl.pathname;
41
42         switch (path) {
```

```
43              case camDataPath:  // a camera wants to register
44              var camera_name = params.camera_name || undefined;
45              var req = ws.upgradeReq;
46              var ipAddress = req.headers['x-forwarded-for'] ||
47              req.connection.remoteAddress ||
48              req.socket.remoteAddress ||
49              req.connection.socket.remoteAddress;
50              camConnections.add(ws, camera_name, ipAddress);
51              break;
52              case clientDataPath:  // a client wants to register to a camera
53              case "/":
54              var camera_name = params.camera_name || "camera0";
55              camConnections.addClientToCamera(camera_name, ws, function(err){
56                  if(err){ws.terminate();}
57              });
58              break;
59              default:
60              console.log("rejected: no valid path");
61              ws.terminate();
62              return;
63          }
64      });
65
66      server.listen(port, ip);
67  }
68
69
70  function verifyClient(info) {
71      var acceptHandshake = false;
72      var accepted = "rejected: no valid password, use 'pass' parameter in the
            handshake please";
73      var ip = info.req.connection.remoteAddress;
74      var clientUrl = url.parse(info.req.url, true);
75      params = clientUrl.query;
76
77      acceptHandshake = params.pass == PASSWORD;
78
79      if (acceptHandshake) {
80          accepted = "accepted";
81      }
82      console.log("new client %s: %s", accepted, info.req.url);
83      return acceptHandshake;
84  }
```

Listing 16: Main file

```
1  var WebSocket = require('ws');
2
3  /** A class that hold WebSocket clients for a camera
```

```
4  * @class
5  */
6  function ClientConnections() {
7      this.clients = [];
8  }
9  /** Lenght of the internal array of clients
10 * @method
11 */
12 ClientConnections.prototype.getLength = function(){
13     return this.clients.length;
14 };
15 /**@method
16 * @param {WebSocket} conn - client websocket connection to add
17 */
18 ClientConnections.prototype.add = function (conn) {
19
20     this.clients.push(conn);
21     // console.log("client connections:%d", this.clients.length)
22 };
23 /**@method
24 * @param {WebSocket} conn - websocket connection to close
25 */
26 ClientConnections.prototype.close = function (conn) {
27     if(this.clients.length < 1){
28         return;
29     }
30     var indx = this.clients.indexOf(conn);
31     this.clients.splice(indx, 1);
32 };
33
34 /**send message to all websockets in the array
35 * @method
36 * @param {string} message
37 */
38 ClientConnections.prototype.sendToAll = function (message) {
39     this.clients.forEach(function (client, ind, arr) {
40         checkSocketOpen(client, function(){
41             client.send(message);
42         });
43     });
44 };
45
46 /**Close all clients in the array
47 * @method
48 */
49 ClientConnections.prototype.closeAll = function (message) {
50     this.clients.forEach(function (client, ind, arr) {
51         try{
```

```
52              client.close();
53          }catch(err){
54              console.error(err.message);
55          }
56      });
57  };
58
59
60  /**A class that holds CameraClient objects
61  * @class
62  */
63  function CameraConnections() {
64      /** @member {Array} - this an array of clientcameras, no websockets
            connections */
65      this.cameras = [];
66  }
67
68  /**
69  * @method
70  * @return {number} - number of cameras in the connected
71  */
72  CameraConnections.prototype.count = function() {
73      return this.cameras.length;
74  };
75
76
77  /**
78  * @method
79  * @return {array} - array of names of the cameras
80  */
81  CameraConnections.prototype.getInfo = function() {
82      var cams = [];
83      this.cameras.forEach(function(element, index) {
84          var name = element.name;
85          if (element.name === undefined){
86              name = "camera"+index;
87          }
88          var infoObject = {name:name, ip:element.ip};
89          console.log(infoObject)
90          cams.push(infoObject);
91      }, this);
92      return cams;
93  };
94
95  /**
96  * @method
97  * @param {WebSocket} conn - connection to add
98  * @param {string} name - name of the socket
```

```
99  */
100 CameraConnections.prototype.add = function (conn, name, ip) {
101     var cname = name;
102     var self = this;
103     if (!cname) {
104         cname = undefined; //name will be based on index
105     }
106 
107     // defining the callbacks for this camera
108     conn.on('message', incomingFromCamera);
109     conn.on('close', closingCamera);
110 
111     var camera = new Camera(conn, cname, ip);
112     this.cameras.push(camera);
113 
114     /** Callled when a connection to a camera is closed
115     * @callback */
116     function closingCamera(code, message) {
117         try{
118             camera.clients.closeAll();
119             self.removeCamera(camera);
120         }catch(err){
121             console.error("Error on closing clients:"+err.message);
122         }
123         console.log("Camera %s closing connection: %d, %s", camera.name, code,
                message);
124     }
125     /** Called when data from a camera is comming
126     * @callback */
127     function incomingFromCamera(message, flags) {
128         try {
129             camera.clients.sendToAll(message);
130         } catch (e) {
131             console.error(e);
132         }
133     }
134     return camera;
135 };
136 /**
137 * @method
138 * @param {(string|object|function)} cameraName - can be the name of the camera or
        a Camera object
139 * @param {WebSocket} clientConn
140 * @param {} callback
141 */
142 CameraConnections.prototype.addClientToCamera = function (cameraName, clientConn,
        callback) {
143     if(typeof cameraName === 'string'){
```

```
144             this.getCamera(cameraName, tryAddClientToCamera);
145         }else{
146             tryAddClientToCamera(cameraName);
147         }
148 
149         function tryAddClientToCamera(camera){
150             if(!camera){
151                 var err = new Error("cannot add client to invalid camera: "+camera);
152                 callback(err);
153                 return;
154             }
155 
156             // new client Callbacks
157             clientConn.on('message', incomingFromClient);
158             clientConn.on('close', closingClient);
159             camera.clients.add(clientConn);
160             callback();
161 
162             /** Called when a client sent data
163             * @callback
164             * @param {string} message
165             * @param {object} flags
166             */
167             function incomingFromClient(message, flags) {
168                 camera.sendMessage(message);
169             }
170             /** Called when a connection to a client is closed
171             * @callback
172             * @param {number} code
173             * @param {string} message
174             */
175             function closingClient(code) {
176                 camera.clients.close(clientConn);
177                 console.log("Closing client connection for: %s camera. info: %d, %s",
                        camera.name, code);
178             }
179 
180         }
181 };
182 
183 CameraConnections.prototype.removeCamera = function(cameraClient){
184     var indx = this.cameras.indexOf(cameraClient);
185     this.cameras.splice(indx, 1);
186 };
187 
188 CameraConnections.prototype.close = function(camera){
189     // this will trigger closingCamera callback
190     this.removeCamera(camera);
```

```
191 };
192 /**
193 * @method
194 * @param {string} name - name of the camera
195 * @param {} callback - callback which receives the camera object
196 */
197 CameraConnections.prototype.getCamera = function (name, callback) {
198     var cameraFound;
199     this.cameras.forEach(function(c, index) {
200         if (c.name == name || name == "camera"+index) {
201             cameraFound = c;
202         }
203     }, this);
204 
205     callback(cameraFound);
206 };
207 
208 
209 /**@class
210 *  A camera client, it has a list of clients attached, and a unique name
211 *  @param {WebSocket} conn - connection object
212 *  @param {String} name - name of this camera (for identification)
213 *  @param {String} ip - ip address of this camera
214 */
215 function Camera(conn, name, ip) {
216     this.conn = conn;
217     this.name = name;
218     this.ip = ip;
219     this.clients = new ClientConnections();
220 }
221 
222 Camera.prototype.sendMessage = function (message) {
223     var conn = this.conn;
224     checkSocketOpen(conn, function(){
225         conn.send(message);
226     });
227 };
228 
229 exports.ClientCamera = Camera;
230 exports.ClientConnections = ClientConnections;
231 exports.CameraConnections = CameraConnections;
232 
233 /**
234 *
235 * @param {WebSocket} socket
236 * @param {} callback
237 */
238 function checkSocketOpen(socket, callback){
```

```
239     if(!socket){
240         console.error('socket does not exists');
241         return;
242     }
243     if(socket.readyState == WebSocket.OPEN){
244         callback();
245     }
246 }
```

Listing 17: implementation of server-side classes

```
1  *** Variables ***
2  ${password}                "password here"
3  ${url}                     localhost:8080
4  ${camera_name_param}       camera_name=
5  ${url_params}              ?pass=${password}
6  ${uri_client}              ws://${url}/client${url_params}&${camera_name_param}
7  ${uri_camera}              ws://${url}/camera${url_params}&${camera_name_param}
8  ${cloud_path}               ../
9  ${cloud_app}                npm start --prefix  ${cloud_path}
10 ${outf}                    log/stdout.txt
11 ${errf}                    log/stderr.txt
12
13
14 *** Settings ***
15 Library           lib/WebsocketLibrary.py
16 Library           OperatingSystem
17 Library           Process
18 Library           HttpLibrary.HTTP
19 Suite Setup       Run Cloud
20 Suite Teardown    Close Cloud
21 Test Teardown     sleep  200 ms
22
23 *** Keywords ***
24 Close cloud
25     Terminate All Processes
26 Run Cloud
27     Remove Files               ${outf}  ${errf}
28     Set Environment Variable   WS_PASSWORD  30022
29     Start Process              ${cloud_app}   alias=cloud_process  stdout=${outf}
           stderr=${errf}  shell=True
30     sleep   1
31     ${is_running} =            Is Process Running  handle=cloud_process
32     Should Be True             ${is_running}   msg=Cloud is not running
33 Wait To Receive Message
34     [arguments]   ${socket}  ${message}
35     Wait Until Keyword Succeeds  5x  5 ms  Receive Next Message   ${socket}  ${
           message}
36 Wait Until Queue
```

```
37      [arguments]   ${socket}   ${n}
38      Wait Until Keyword Succeeds   5x  50 ms  Messages In Queue Should Be   ${
            socket}   ${n}
39  Create Camera Socket
40      [arguments]   ${name}
41      create socket  ${name}  ${uri_camera}${name}
42      Create Camera Socket Noname
43      create socket  noname  ${uri_camera}
44  Create Client Socket
45      [arguments]   ${name}   ${camera_socket}
46      create socket  ${name}  ${uri_client}${camera_socket}
47  Random Message
48      ${randint}  Evaluate   str(random.randint(0, sys.maxint))   modules=random,
            sys
49      [Return]   ${randint}
50  Send Random Message From
51      [arguments]  ${socket}
52      ${message}   Random Message
53      Send From Socket  ${socket}  ${message}
54  Get Cameras
55      Create Http Context   ${url}   http
56      GET   /cams
57      Response Status Code Should Equal   200
58      ${body} =   Get Response Body
59      Should Start With   ${body}   {
60      Log Json  ${body}
61  Number Of Cameras Should Be
62      [arguments]   ${n}
63      Create Http Context   ${url}   http
64      GET   /cams
65      Response Status Code Should Equal   200
66      ${body} =   Get Response Body
67      Json Value Should Equal  ${body}  /count  ${n}
68      Log Json  ${body}
69
70
71      *** Test Cases ***
72      close sockets
73      create Camera Socket         camera0
74      Create Client Socket         client          camera0
75      sleep                               100 ms
76      Close Socket                 client
77      Create Client Socket         client2          camera0
78      sleep                               100 ms
79      Close Socket                 client2
80
81
82      Check Socket Library works
```

```
83      Create Socket                client0  ${uri_camera}
84      Do Exist Socket             client0
85
86      Create camera client and send
87      Create Camera Socket         camera0
88      Create Client Socket         client0  camera0
89      send From Socket             camera0  hi
90      send From Socket             client0  hello
91
92      cameras without name and with name
93      create Camera Socket Noname
94      create Camera Socket Noname
95      create Camera Socket Noname
96      create Camera Socket         1111special-cam1111
97      create Camera Socket Noname
98      create Camera Socket Noname
99      sleep  80 ms
100     Number Of Cameras Should Be    6
101
102
103     5 cameras, 1 client, 5 messages
104     create Camera Socket         camera0
105     create Camera Socket         camera1
106     create Camera Socket         camera-special
107     create Camera Socket         camera3
108     create Camera Socket         camera4
109     Create Client Socket         client          camera-special
110     sleep                                100 ms
111     Get Cameras
112
113     Send Random Message From             camera-special
114     Send Random Message From             camera-special
115     Send Random Message From             camera-special
116     Send Random Message From             camera-special
117     Send Random Message From             camera-special
118     Wait Until Queue          client   5
119
120
121
122     several cameras with several clients, bidirectional communication
123     Create Camera Socket        cam
124     Create Camera Socket        camf
125     Create Client Socket        client       cam
126     Create Client Socket        client1      cam
127     Create Client Socket        client2      camf
128     Create Client Socket        client3      camf
129     sleep                                100 ms
130     Get Cameras
```

```
131
132     Send Random Message From             cam
133     Send Random Message From             camf
134
135     Wait Until Queue          client    1
136     Wait Until Queue          client1   1
137     Wait Until Queue          client2   1
138     Wait Until Queue          client3   1
139
140     Send Random Message From             client
141     Send Random Message From             client1
142     Send Random Message From             client2
143     Send Random Message From             client3
144
145     Wait Until Queue          cam       2
146     Wait Until Queue          camf      2
147
148     Http Server
149     Get Cameras
```

Listing 18: RobotFramework test file

```
1  #!/usr/bin/python
2  import subprocess
3  import sys
4  from Queue import Queue, Empty
5  from threading import Thread
6  import websocket
7  from robot.api import logger
8
9  __version__ = '0.1'
10 __author__ = "Alvaro Bolanos Rodriguez"
11
12
13 class WebsocketLibrary:
14     ROBOT_LIBRARY_SCOPE = 'TEST CASE'
15     ROBOT_LISTENER_API_VERSION = 2
16
17     def __init__(self):
18         self.ROBOT_LIBRARY_LISTENER = self
19         # self.url = "%s:%d" % (host, port)
20         self.socketDic = {}
21
22     def _start_suite(self, name, attrs):
23         print 'started suite'
24
25     def _end_suite(self, name, attrs):
26         print 'Suite %s (%s) ending.' % (name, attrs['id'])
27
```

```
28      def _start_test(self, name, attrs):
29          pass
30          def _end_test(self, name, attrs):
31          self.stop_all_sockets()
32
33      def _get_socket(self, name):
34          try:
35              return self.socketDic.get(name)
36          except Exception as e:
37              logger.error(e.message)
38
39      raise Exception("%s socket not found in list" % name)
40
41      def create_socket(self, name, uri):
42          ws = self.WebSocketConnection(uri, name=name)
43          self.socketDic[name] = ws
44          logger.info("created %s using %s" % (name, uri))
45
46      def do_exist_socket(self, name):
47          if self.socketDic.has_key(name):
48              logger.info("'%s' exists" % name)
49          else:
50              raise AssertionError("'%s' does not exist" % name)
51
52      def close_socket(self,name):
53          s = self._get_socket(name)
54          s.stop()
55
56      def send_from_socket(self, socket, message):
57          try:
58              s = self._get_socket(socket)
59              s.send_to_socket(message)
60              logger.info("%s is sending '%s' message" % (s.name, message) )
61          except websocket.WebSocketConnectionClosedException as e:
62              logger.warn(e.message + ".Try using 'Wait to' keyword style")
63
64      def stop_all_sockets(self):
65          for name, socket in self.socketDic.items():
66          socket.stop()
67          self.socketDic = {}
68
69      def receive_next_message(self, name, expected):
70          ws = self._get_socket(name)
71          try:
72              received_message = ws.receive_next_message()
73          except Empty as e:
74              msg = e.message+"Message is not in Queue yet"
75              logger.warn(msg)
```

```
76              raise AssertionError(msg)
77          if not ws:
78              raise AssertionError("there is no websocket")
79              if not received_message == expected:
80                  msg = "Messages do not match:'%s' is not '%s'" % (received_message
                        , expected)
81                  logger.warn(msg)
82                  raise AssertionError(msg)
83
84      def messages_in_queue_should_be(self, name, n):
85          expected = int(n)
86          s = self._get_socket(name)
87          actual = s.in_queue.qsize()
88          if actual != expected:
89              raise AssertionError("number of elements does not match, was %d,
                    expected %d" % (actual, expected))
90
91
92  class WebSocketConnection(Thread):
93      def __init__(self, url, name):
94          Thread.__init__(self, name=name)
95          # websocket.enableTrace(True)
96          self.url = url
97          self.name = name
98          self.in_queue = Queue(10)
99          self.ws = websocket.WebSocketApp(self.url,
100         on_message=self.on_message,
101         on_error=self.on_error,
102         on_close=self.on_close,
103         on_open=self.on_open)
104         self.setDaemon(True)
105         self.start()
106
107     def run(self):
108         self.ws.run_forever()
109
110     def on_message(self, ws, message):
111         self.in_queue.put(message)
112         logger.info("from %s:%s" % (self.name, message))
113
114     def on_error(self, ws, error):
115         logger.error(error)
116
117     def on_close(self, ws):
118         logger.info("closed %s" % self.name)
119
120     def on_open(self, ws):
121         logger.info("opened %s" % self.name)
```

```
122
123     def stop(self):
124         self.ws.close()
125
126     def send_to_socket(self, data):
127         self.ws.send(data)
128
129     def receive_next_message(self):
130         return self.in_queue.get(block=False)
```

Listing 19: RobotFramework websocket library