POLITECNICO DI TORINO

Projects and laboratory on communication systems

FISSURE SENSOR:

OPTICAL MOUSE AND FEZ SPIDER II BOARD

FEZ49

Professor:

Albertengo Guido

Students:

Antonietta Simone Domenico (s239869)

Bulgarella Fabio (s237094)

Loro Matteo (s242925)

Index

Introduction

The purpose of this report is to show our project for the *Project and Laboratory on Communication Systems* course, commissioned by Reply Company.

Reply Company proposed four different kind of sensor to realize: heat sources, river water level, critical fissure measurement, temperature/humidity measurement. The one we choose was the third.

Fissure detection and measurement are very important in domestic environments because, monitoring a fissure, you can establish the actual condition of the wall considered.

Our work consists on a board made up by several hardware components, taking these kinds of data from sensors:

- Fissure size (X and Y axes)
- Temperature
- Humidity
- Pressure

Storing them into a SD card and sending a properly set JSON file to the cloud, we are able to provide a database of our measurement, easily available from a website developed by us.

Hardware components

FEZ spider II

The FEZ Spider II is the main board, on which runs *.NET Micro Framework*; we programmed and debugged the firmware of the board from Microsoft Visual Studio using C# language, writing custom classes that instantiate several threads, each one with a specific role.

Technical specifications:

• Processor:	120 MHz 32-bit ARM Cortex-M3
• User RAM:	13,67 MB
• User flash:	2,87 MB
• Operating temperature:	-40°C / +85°C

Optical mouse

To measure X and Y distance we use a TeckNet HYPERTRAK gaming mouse. We chose a gaming mouse because we need to acquire data with a defined frequency and without losing any movements. A standard mouse would have not been a good choice because it suffers of the “stand-by problem”, a power save modality that decrease sensor sensibility and responsiveness, resulting on a movements loss when mouse is moved again and sensor has to be woken up. In our mouse standby state is disabled.

Technical specification:

• Mouse sensor:	ADNS-9800
• Resolution:	8200 dpi
• Acquisition time:	8 ms
• Connection:	USB cable
• Price:	23,00 €

To evaluate the resolution of the mouse in mm: 1 inch / 8200 dpi = 25,4 mm/ 8200 dpi = 0,003 mm/dot. In our case the useful resolution is 0,01 mm due mechanical vibrations on wall and noise, they can cause a wrong measurement, so the accuracy is set to the value specified.

Moreover, by test made with the standard Mouse class of the FEZ Spider SDK, we noticed that values obtained from our mouse were wrong and not accurate (even with different precision in the two axis), so we decided to write a brand-new mouse driver. We used the RawDevice class available on the FEZ SDK to have a direct access to the USB bus and read raw data sent in polling by the device to the board. After a first studying phase of the HID protocol, we were able to extract correctly the delta variations (of the two axis) from the received packet in order to provide the right measurements, consistent with mouse sensor specification.

It is worth to notice that displacement coordinates are saved every 6 seconds in a file named “MouseData” inside persistent storage, so we are able to restore them at boot time just in case a power failure takes place.

Temperature-pressure-humidity sensor

We used Bosch BME280 sensor to measure ambient temperature, pressure and humidity. In real fissure measurement, it is useful to have ambient parameters to evaluate correctly variations caused by ambient variations or by real movement in the wall and to evaluate how thermal cycles affect the fissure size.

Technical specifications:

• Temp range:	0°C / 65°C
• Temp precision:	0,2°C
• RH range:	0% / 100%
• RH precision	3%
• Press range:	300hPa / 1100hPa
• Press precision:	1hPa
• Interface:	I2C / SPI
• Supply voltage:	1,7V / 3,6V
• Price:	5,00 €

We connected the sensor using the I2C interface integrated into the FEZ Spider; we acquire the measurements just one time every 2 minute because sensor driver internally performs more reading wich are consequently averaged, reducing noise. To communicate with the sensor, we use the I2C library integrated into the .NET Microframework environment and the sensor driver, optimized by us to manage the *Out of Range* feature and to align compensation formulas to the calibrations made (shown below).

The sensor was calibrated in Politecnico LED, measuring temperature and humidity provided by Greisinger GFTH95 thermo-hygrometer and providing a software compensation of sensor offsets.

The GFTH95 was calibrated into a climatic chamber contained inside the DET department of the Politecnico, which was calibrated by an INRIM instrument, so it is reasonable to consider the test accurate.

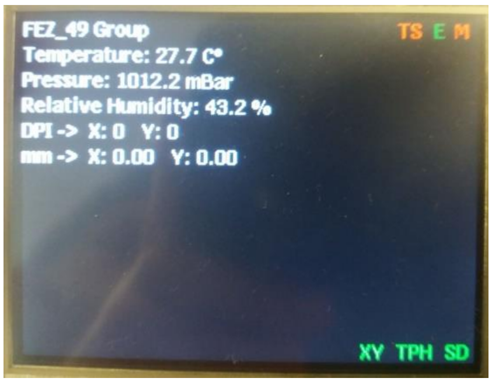
LCD display

We used 240x320 LCD display to show essentials information about our board:

- the measure provided by the sensors

- the connection of all the components of the system

As you can see in the image below, display shows last measurements and all information about internet, sensors and components connection state, using two different colors to show if they are connected or disconnected (green and red):



```
FEZ_49 Group                    TS E M
Temperature: 27.7 C°
Pressure: 1012.2 mBar
Relative Humidity: 43.2 %
DPI -> X: 0  Y: 0
mm -> X: 0.00  Y: 0.00

                            XY TPH SD
```

• XY:	Mouse
• TPH:	BME280
• SD:	SD card
• TS:	Time Service
• E:	Ethernet
• M:	MQTT

Other components

The other components used are:

- SD card: to store JSON files that have to be sent to AWS IoT platform.
- Ethernet connector: to provide Internet connection to the board, sharing it from a computer to the microcontroller.
- Button: to reset the mouse position during the execution; it is useful during tests and installation.

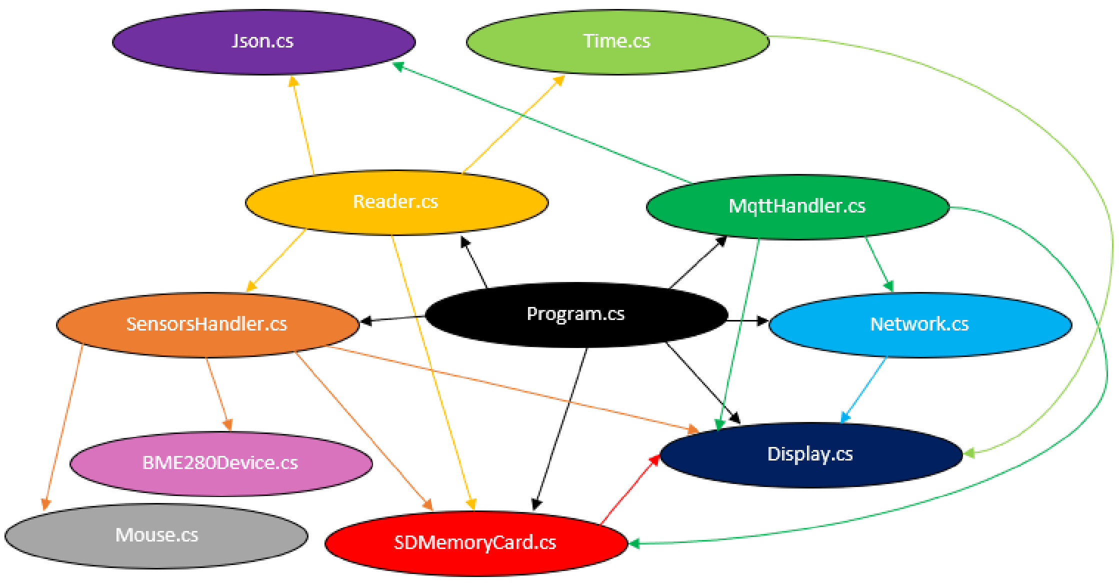
FEZ firmware

Classes developed

The libraries written by us are:

- Program.cs: main class in which is contained the bootstrap code
- Reader.cs: it is used to read the measurements provided by the sensors, enveloping them in a file
- SensorsHandler.cs: it manages the sensors and creates measurements data
- SDMemoryCard.cs: it provides methods to read and write the SD Card
- Display.cs: it contains all the functions useful to represent some data on the LCD screen
- Network.cs: it manages the Ethernet connection
- MqttHandler.cs: it is used to communicate with the remote server using MQTT protocol
- Json.cs: library that creates the JSON file, according to the format used
- Time.cs: it manages the data and time information on the board
- Mouse.cs: it manages the mouse sensor
- BME280Device.cs: it manages the BME280 sensor

This is a graph on which is represented the way in which these libraries are connected:



Power on

When the board is turned on, it initializes:

- Mouse: check connection, load last displacement value present in mouse data file.
- TPH sensor: check connection, internal start up procedure.
- SD card: check connection, look for not synchronized (in time) JSON files and remove them.
- Network: initial configuration, check connectivity.
- Time service: enable time synchronization thread.

Data and Time management

Due to the fact that FEZ is not time synchronized at boot, a class has been introduced to handle time related problems. A thread in charge of correctly synchronize time is started at boot and when connectivity becomes available it starts the synchronization process:

1. Resolve NTP hostname by a DNS request
2. Configure TimeService with ntp ip address and start it
3. Wait for SystemTimeChanged event that notify the correct time synchronization; it also saves the time offset between old time and new synched time (Time.FirstSyncTimeOffset) so we can update already acquired not synched measurements.

When a new JSON file is written on SDcard, the boolean value Time.IsTimeSynchronized is checked: if it's false a suffix, containing timestamp in "Ticks", is added to file name before saving it on persistent storage (20110601T003320_ 1351693418327).

When will be the moment to send that file to MQTT Gateway, his name will be checked and if it contains the timestamp ticks suffix, a conversion of measurement dates to synchronized ones will be started. The conversion method uses Time.FirstSyncTimeOffset as indeed an offset that has to be added to the old unsynched datetimes present inside the JSON file.

JSON format

We create the JSON file using the class Json.cs developed by us. This class contains functions that, passing the measurements as argument, return the array of bytes used to generate the file to be stored. The time is taken by the date and time information stored in the microcontroller and updated by the class *Time.cs*, written by us.

This is an example of a JSON file produced:

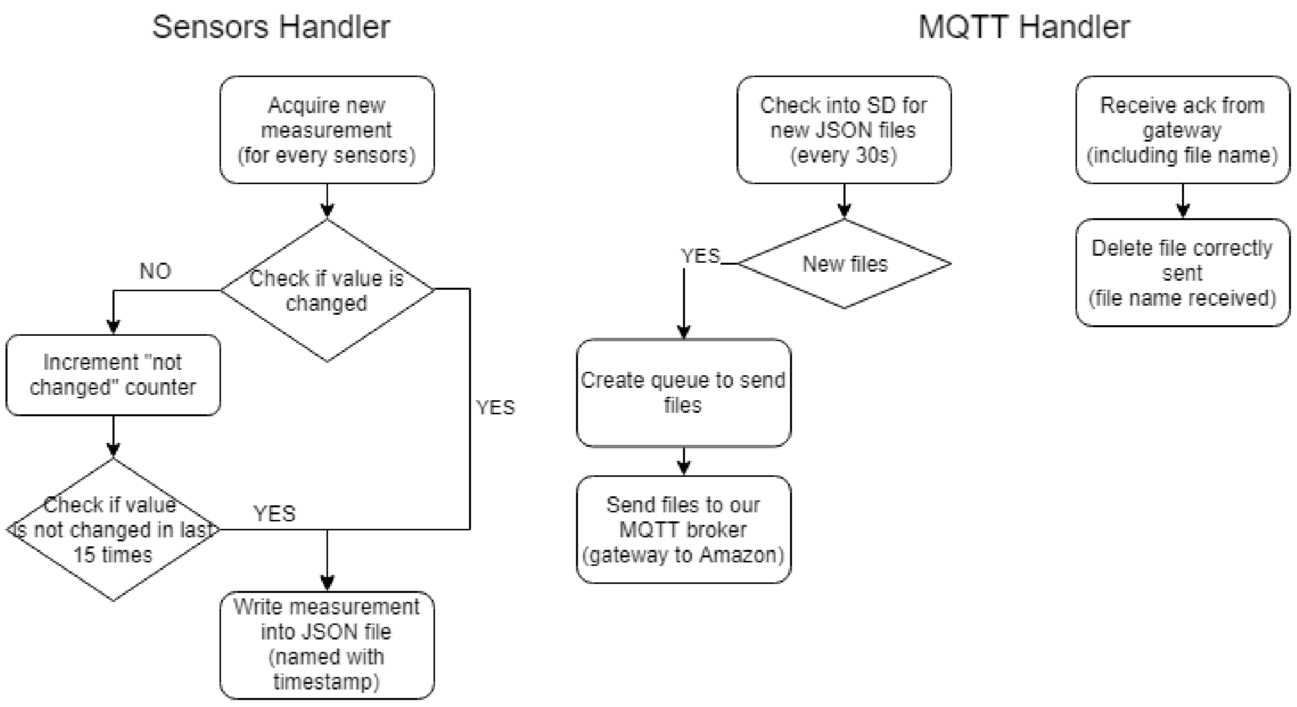
```
{ "version": 2, "device_id": "FEZ_49",  "iso_timestamp": "2018-07-23T08:35:20+00:00",  "measurements":
[
   { "sensor_id": 1, "iso_timestamp": "2018-07-23T08:35:20+00:00", "value": 0.11, "status": "OK" },
   { "sensor_id": 2, "iso_timestamp": "2018-07-23T08:35:20+00:00", "value": 0.06, "status": "OK" },
   { "sensor_id": 3, "iso_timestamp": "2018-07-23T08:35:20+00:00", "value": 27.9,"status": "OK" },
   { "sensor_id": 4, "iso_timestamp": "2018-07-23T08:35:20+00:00", "value": 1012.3,"status": "OK" },
   { "sensor_id": 5, "iso_timestamp": "2018-07-23T08:35:20+00:00", "value": 41.9,"status": "OK" }
]}
```

Storage and submission of data

The data are acquired every 2 minutes; the software, for each kind of measurement, controls if the value is equal to the previous one. In positive case, it increments a counter and, only if this counter is equal to 15, prints this measure on the file and reset the counter. In negative case, it prints directly measure on the file (resetting the counter also this time). In this way, the file contains only the measurements that are recently changed or a measure that has not experienced changes in the last 30 minutes.

If there are problems with a sensor, that, for instance, was disconnected or failed, the software sets the relative value to 0 and the status to FAIL. Also in this case, the information is printed into JSON file following the same previous algorithm.

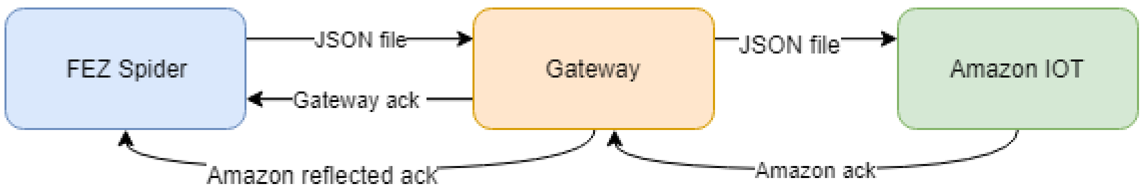
The algorithm implemented to manage the SD card is made by 2 threads, one for the writing of new files, the other for the sending and removing operation of the files stored:


Sensors Handler
MQTT Handler
Acquire new measurement (for every sensors)
Check if value is changed
NO
YES
Increment "not changed" counter
Check if value is not changed in last 15 times
YES
Write measurement into JSON file (named with timestamp)
Check into SD for new JSON files (every 30s)
New files
YES
Create queue to send files
Send files to our MQTT broker (gateway to Amazon)
Receive ack from gateway (including file name)
Delete file correctly sent (file name received)

The SD card mounted has got a 2GB storage capacity, so, considering that a file stored on it takes 4kB (due two disk block size), the maximum number of files that can contain is: 2GB/4kB=512000. Considering that the frequency on which files are stored is 1 file each 2 min, the system can store files without saturate the SD card for: 512000*2 = 1024000 min = 17066,67 h = 711.1 days = 1.95 years. Considering that the algorithm deletes the files sent to the server, the system can stay about 2 years without Internet connection. In case of production of this device, it could be possible to use an integrated non-volatile memory, as a flash, also with less storage capacity.

Cloud connection

File transmission between FEZ and Amazon IoT is done using an intermidiate gateway, hosted on a PC connected directly to the board. The following scheme explain how the entire communication takes place:



MQTT protocol has been used for both communication phases: Fez to Gateway and Gateway to Amazon IoT. To achieve this result two different libraries have been adopted in our project:

- M2MQTT -> used on FEZ as MQTT client to Gateway
- MQTTnet -> used on Gateway as MQTT Broker for FEZ and as MQTT Client to Amazon IoT Broker

Transmission algorithm:

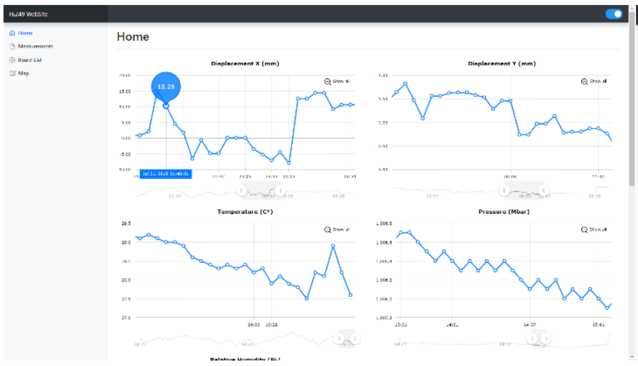
1. JSON file is transmitted from FEZ to GW (with MQTT QoS set to exactly once).
2. GW sends an Ack toward FEZ to notify that JSON file was taken in charge.
3. GW (that has a queue with JSON file to be sent) transmits the measurements acquired to Amazon IoT (with MQTT QoS set to at least once).
4. Amazon IoT sends an Ack toward GW to notify that the JSON file was correctly received and measurements processed correctly.
5. Finally, GW build an Ack containing the filename of correctly sent JSON and send it to FEZ.
6. FEZ process incoming Ack and delete corresponding file from SDcard.

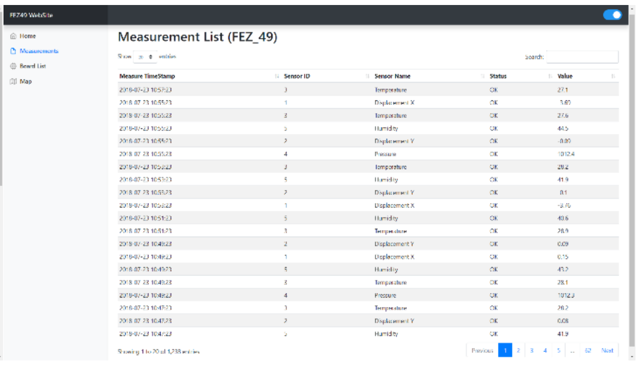
Complex Website

The website is written using PHP, CSS, HTML, JavaScript and the Amazon PHP SDK. It is divided into 4 pages:

- Index: home page of the site, containing the graphs of the measurements of our group
- Measurements: page that contains the table of all the measurements of the group
- Boards: it contains the list of the board available on the server; if you click on one of them, the relative measurements page is displayed
- Map: it displays a map on which you can see the geographical position of all the sensors available on the server

These are screens of the pages described:



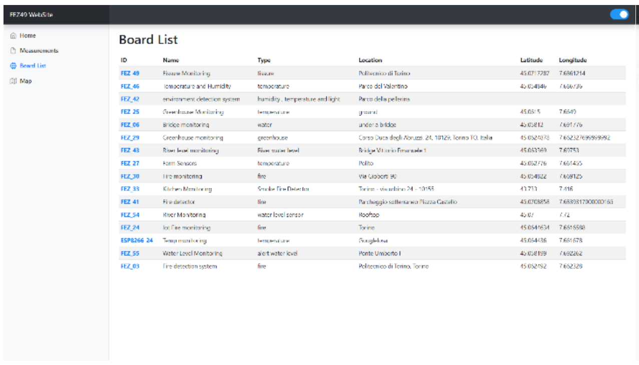


Measurement List (FEZ_49)

Show 20 entries Search:

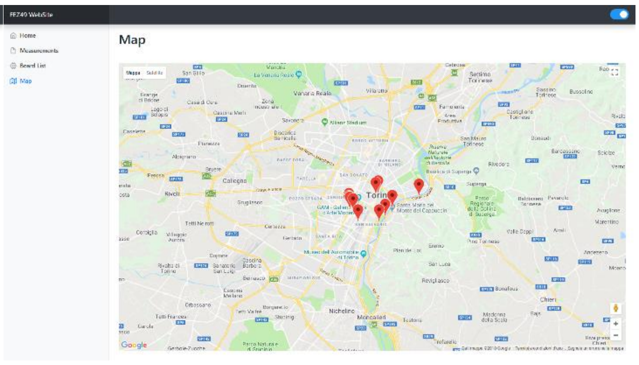
Measure TimeStamp	Sensor ID	Sensor Name	Status	Value
2018-07-23 10:57:23	3	Temperature	OK	27.1
2018-07-23 10:55:23	1	Displacement X	OK	3.69
2018-07-23 10:55:23	3	Temperature	OK	27.6
2018-07-23 10:55:23	5	Humidity	OK	44.5
2018-07-23 10:55:23	2	Displacement Y	OK	-0.09
2018-07-23 10:55:23	4	Pressure	OK	1012.4
2018-07-23 10:53:23	3	Temperature	OK	28.2
2018-07-23 10:53:23	5	Humidity	OK	41.9
2018-07-23 10:53:23	2	Displacement Y	OK	0.1
2018-07-23 10:53:23	1	Displacement X	OK	-3.76
2018-07-23 10:51:23	5	Humidity	OK	40.6
2018-07-23 10:51:23	3	Temperature	OK	28.9
2018-07-23 10:49:23	2	Displacement Y	OK	0.09
2018-07-23 10:49:23	1	Displacement X	OK	0.15
2018-07-23 10:49:23	5	Humidity	OK	43.2
2018-07-23 10:49:23	3	Temperature	OK	28.1
2018-07-23 10:49:23	4	Pressure	OK	1012.3
2018-07-23 10:47:23	3	Temperature	OK	28.2
2018-07-23 10:47:23	2	Displacement Y	OK	0.08
2018-07-23 10:47:23	5	Humidity	OK	41.9

Showing 1 to 20 of 1,238 entries Previous 1 2 3 4 5 … 62 Next



Board List

ID	Name	Type	Location	Latitude	Longitude
FEZ_49	Fissure Monitoring	fissure	Politecnico di Torino	45.0717287	7.6861214
FEZ_46	Temperature and Humidity	temperature	Parco del Valentino	45.054846	7.686736
FEZ_42	environment detection system	humidity , temperature and light	Parco della pellerina		
FEZ_25	Greenhouse Monitoring	temperature	ground	45.0615	7.6649
FEZ_06	Bridge monitoring	water	under a bridge	45.03812	7.691776
FEZ_29	Greenhouse monitoring	greenhouse	Corso Duca degli Abruzzi, 24, 10129, Torino TO, Italia	45.0624078	7.65232769999992
FEZ_43	River level monitoring	River water level	Bridge Vittorio Emanuele 1	45.063369	7.69753
FEZ_27	Farm Sensors	temperature	Polito	45.062776	7.661455
FEZ_30	Fire monitoring	fire	Via Gioberti 90	45.054822	7.668125
FEZ_33	Kitchen Monitoring	Smoke Fire Detector	Torino - via urbino 24 - 10155	43.713	7.416
FEZ_41	Fire detector	fire	Parcheggio sotterraneo Piazza Castello	45.0708858	7.68393170000163
FEZ_54	River Monitoring	water level sensor	Rooftop	45.07	7.72
FEZ_24	Iot Fire monitoring	fire	Torino	45.0644634	7.6616568
ESP8266_24	Temp monitoring	temperature	Googlekea	45.064436	7.661678
FEZ_55	Water Level Monitoring	alert water level	Ponte Umberto I	45.058199	7.692262
FEZ_03	Fire detection system	fire	Politecnico di Torino, Torino	45.062492	7.662320



Test and simulation

To simulate the behavior of a fissure, we built a mechanical fissure simulator: this is a device made up by two mechanical slices capable to move each one in a certain direction. In this way we can obtain the same behavior of a fissure that extends or reduce its size in both the planar directions. The plane on which the mouse moves is covered to millimeter paper, so it is simple to check visibly if the value acquired during the tests are right. We made tests of some hours to test the correctness of the project, from the acquisition of the sensors to the communication to the server, including also the displaying of the measurements on the website.