4th Year IoT Project  
Aquarium Monitoring System: HCFF

Julien Rose, Achille Bayart, Nicolas Rigaudy

Table of contents

[Introduction and context 3](#_Toc44353197)

[Completed work 3](#_Toc44353198)

[Hardware 3](#_Toc44353199)

[Raspberry Pi 3](#_Toc44353200)

[Temperature 4](#_Toc44353201)

[Brightness 6](#_Toc44353202)

[Image Recognition 8](#_Toc44353203)

[Initial Idea 8](#_Toc44353204)

[OpenCv 8](#_Toc44353205)

[Android App 10](#_Toc44353206)

[Figma 11](#_Toc44353207)

[12](#_Toc44353208)

[12](#_Toc44353209)

[Android Studio 12](#_Toc44353210)

[Data collection and linking 13](#_Toc44353211)

[Rest Client 13](#_Toc44353212)

[Project perspective 13](#_Toc44353213)

[Planned additions 13](#_Toc44353214)

[Challenges 14](#_Toc44353215)

[Conclusion 14](#_Toc44353216)

# Introduction and context

# Completed work

## Hardware

### Raspberry Pi

The Raspberry Pi (RPi) is a nano computer.

Figure . Raspberry Pi 4 Model B

We decided to choose the Raspberry Pi 4 Model B because it is the latest model of Raspberry Pi and it fit perfectly with what we aimed to achieve in this project. We chose a model with 2GB of random-access memory as this was sufficient for our use.

We installed Raspbian (Raspberry Pi OS) which is a free operating system based on Debian, a Linux distribution.

To setup the RPi efficiently, we connected to it:

* **A USB type-C power supply** to provide power to the board.
* **A microSD card**: the nano computer needs an SD card to store all its files and the Raspbian operating system. We selected a microSD of 64 GB to be safe as we knew we would have to store video footage of the camera on the RPi.
* **A computer screen**: this is used to view the Raspbian desktop environment. We used a standard HDMI cable plus a micro-HDMI to HDMI adapter, to connect Raspberry Pi 4 to the screen.
* **A USB keyboard and a USB mouse** to be able to configure and work on the RPi.



Figure . B01 Camera Module

Moreover, we bought a B01 camera module especially for the Raspberry Pi. This camera is used to detect and count the fish in the aquarium tank, using OpenCv image recognition (see page 8).

At first, we tried using a Withings and a Samsung camera for the project as we already had them on hand but, being smart cameras with a lot of features and security, they were very hard to connect to the RPi. Achille’s father normally uses those two cameras to check on the aquarium when he is away but they therefore both already have dedicated apps and it was too complicated for us to integrate them into our prototype.  
We thus decided to use a camera module specially made for the Raspberry Pi, it’s useful and not very expensive.

### Temperature

The first information we wanted to measure was the aquarium’s temperature as it is very important for the fish’s healthcare, especially certain species that need a precise temperature-controlled environment.

Figure . DS18B20 temperature sensor

We decided to choose the DS18B20 sensor which is very commonly used in conjunction with the Raspberry Pi. This sensor has a measurement range from -55°C to +125°C and it is waterproof.

We started our tests with only one temperature sensor. The temperature sensor was quite easy to connect and read from as we used a preexisting Python library to control it. We also took the result of the temperature measures and saved them into a text file.

We then added another temperature sensor to the prototype breadboard, in order to get a more precise reading of the water temperature in the tank, with the two sensors spread out to opposite corners.

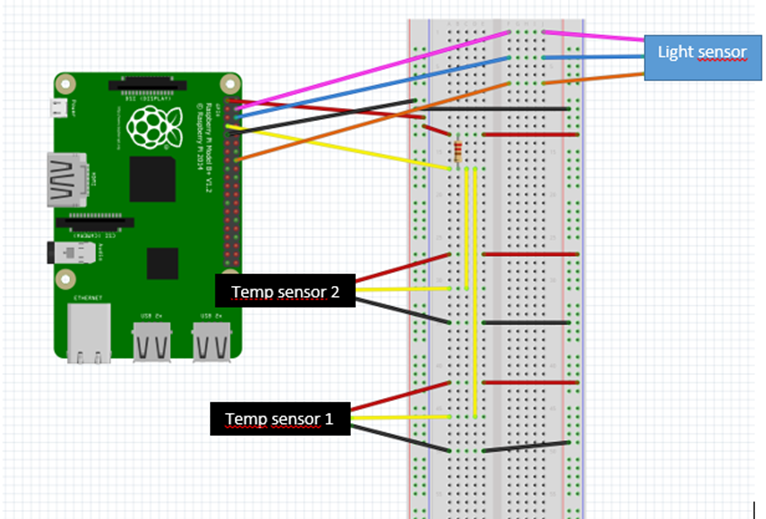


Figure . Prototype board wiring diagram (made using Fritzing)

We used a 4,7 kΩ resistor to connect the sensors and the cabling is done as follows:

* red wire to pin 1 (3,3V power)
* yellow wire to pin 7 (GPIO 4)
* black wire to pin 9 (Ground)

This wiring allowed us to measure the average temperature of the two sensors. We can see below the Python code used to save the average temperature to a text file.

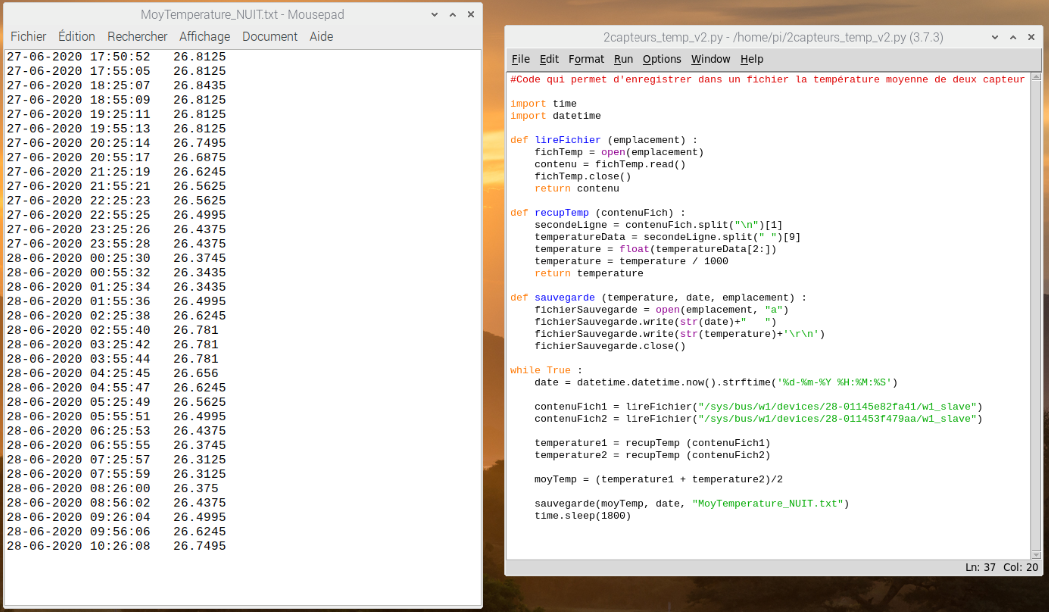


Figure . Python code that saves the average temperature in the "MoyTemperature\_NUIT" text file

The average temperature is saved every 30 minutes (1800 seconds). As you can see on the previous picture, we leave the sensors in the aquarium during one whole night. We have a maximum difference of 0,5°C and the temperature stays around 26,5°C. This is exactly what we expected.

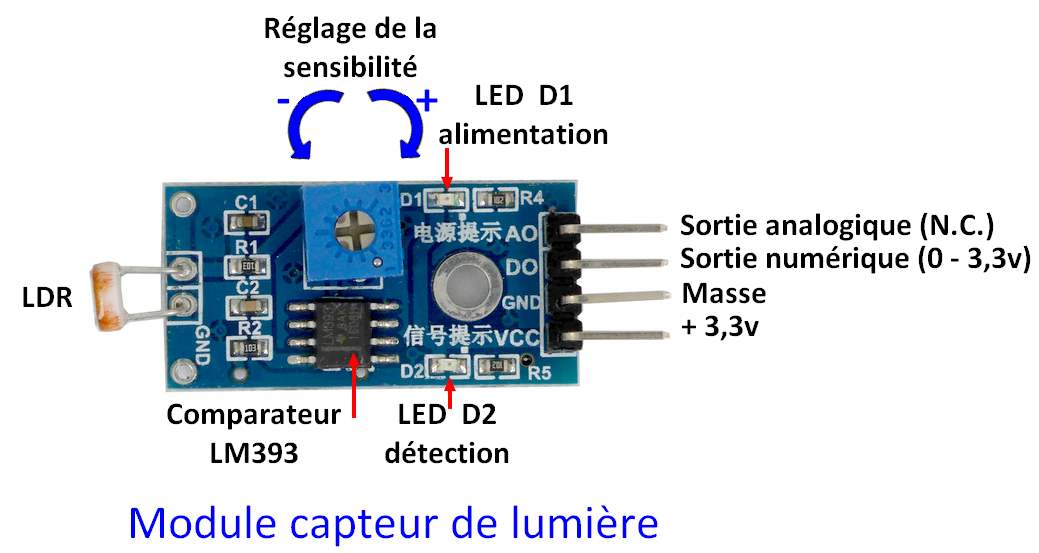


Figure . Full setup of Raspberry Pi with all sensors and camera module

### Brightness

The second information we wanted to get from our device was the brightness inside the aquarium, to be able to check if the tank lights are on or off. We decided to add another sensor to the Raspberry Pi: the LDR (Light Dependent Resistor) light detector module.

This sensor is used by setting a light threshold by adjusting the sensitivity of the LDR. The output of the module then depends on the threshold as explained here:

* Light intensity < threshold: output = 0
* Light intensity > threshold: output = 1

**Power**

**Detection**

**+5V**

**Ground**

**Digital output**

**Analog output**

**Sensitivity adjustment**

**Comparator LM393**

Figure . Light Sensor Module

As you can see in the wiring diagram (see page 4), this sensor is connected as follows :

* pink wire to pin 4 (5V power)
* blue wire to pin 6 (Ground)
* orange wire to pin 16 (GPIO 23)

The python code of the light sensor is below:

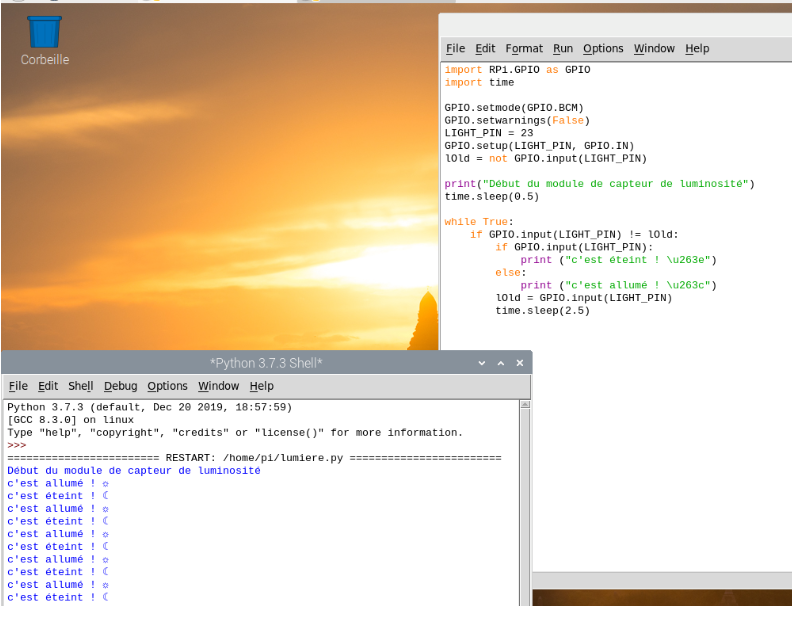


Figure . Brightness sensor Python code

A new line is printed when the brightness changes. When the light intensity is inferior to the fixed threshold, the code prints « it’s off » and when the light intensity is superior to the fixed threshold, the code prints « it’s on ».

## Image Recognition

### Initial Idea

One feature we wanted to work on was image recognition. This was first intended as a way to count and detect the different fish and show them according to their species, also allowing to analyze in detail their behavior to alert of any signs that could warn the user of potential health problems.  
However, to be able to recognize the fish this accurately, we would have needed to use TensorFlow, a very detailed and thus quite complex machine learning platform and use a custom library of data (a large amount of images of the fish with boundary boxes and labelled according to their species). This path was very interesting but could have been a complete AI project, and we did not have the time or resources to go through with this idea in its entirety.

OpenCv  
We thus opted for simpler image recognition, using a Python library called OpenCv, which is a basic yet powerful too that can be used in a variety of ways with images and videos. It enabled us to detect fish movement in short videos and have an approximate count of the number of fish visible on camera during that video.

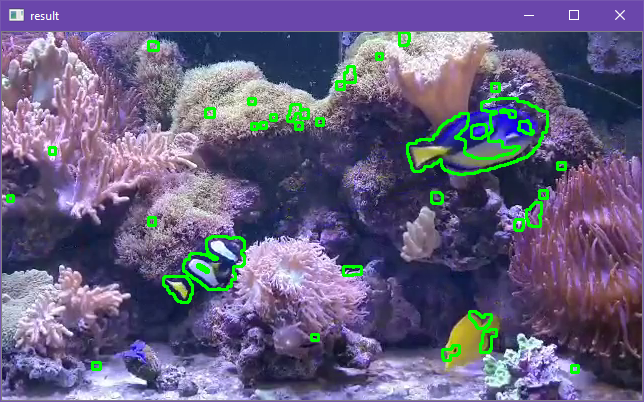


Figure . Detection of contours with movement

In this image, we can see the contours the program detects. This is done by comparing frame by frame a black and white version of the image. This image is also modified in various ways (blurs, thresholds, etc.) to accentuate contrast between shapes to identify the contours more easily. The program looks at contours that change position between two frames and displays them on screen.

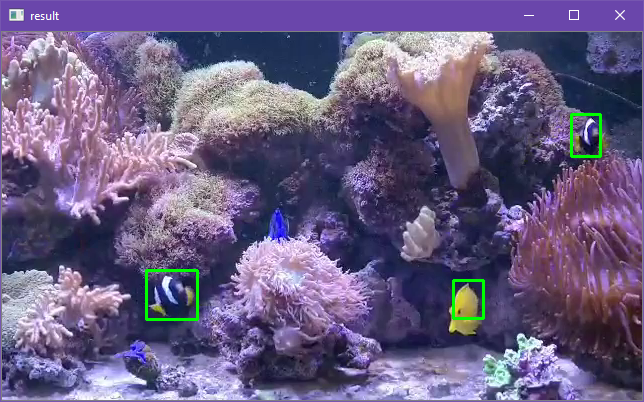


Figure . Drawing boxes from contours

In this second image, we can see that most of the small contours identified in the first step are not recognized as fish and only the fish have boxes drawn around them. This is done by limiting the size of the contours that should get a boundary box to a minimum to eliminate most false positives coming from moving debris or from the anemones.

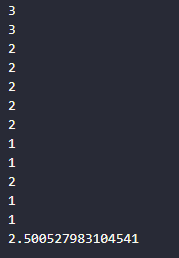


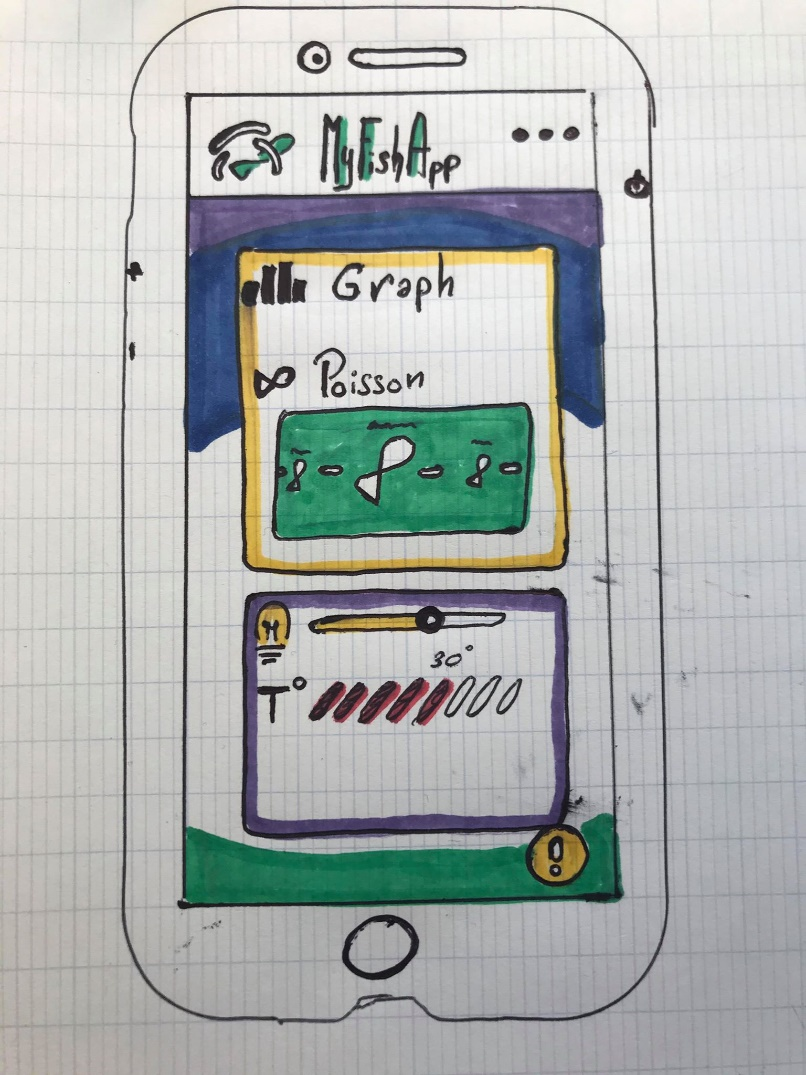
Figure . Number of fish for each frame and average for the video segment

The program can then count the number of boxes present on screen for each frame to have an approximate amount of fish visible by the camera. When the program is exited, an average is calculated of the number of fish present during the whole video segment.

## Android App

Our goal was to be able to check all the information of the aquarium that could affect fish health. We decided to use an Android application as our user interface.

The app needs to show the main elements of information, such as the different measures done by the sensors (temperature, light, etc.) and the last image of the fish. The interface must be designed in a smart way that is intuitive to use and as informative as possible for the user it.

To establish our vision of a “smart interface” we made this first drawing:

The main idea of our app is shown here, with, in the center, a fish information box, where you can choose which fish you want to see and that brings you to a page with more information on the fish.  
Below, there is the main information box, which displays all current sensor readings and brings to different pages of more detailed graphs for each sensor.

### Figma

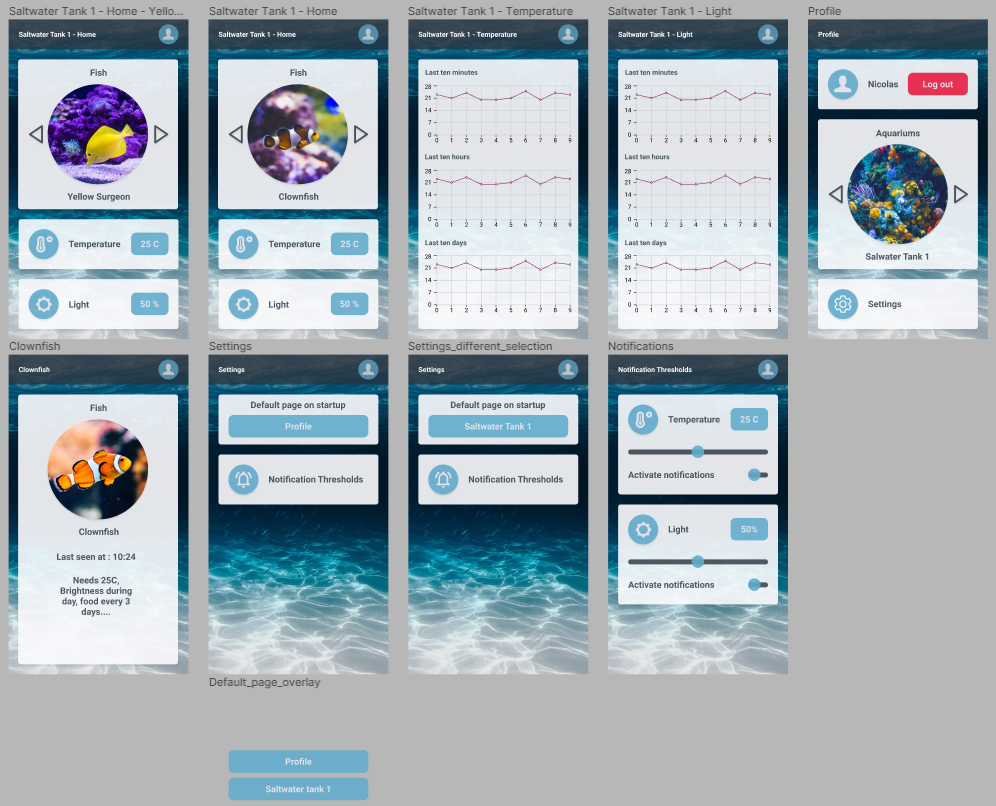
To design our interface, we first used a prototyping software called Figma. Figma helped us a lot to recreate our vision and drawing in a visual prototype. Figma is an interface design application, and thus also gives code for each element of the interface to use in Android Studio. This code was very helpful to understand how android interfaces work and when it is necessary to use smart interface tools.

Figure . Figma App Prototype

In this image, we can see the full prototype of the app, with a home page to select the aquarium tank we want to look at, as well as change the settings and also an option to log out of our profile. We can also see the different pages for individual fish, for the data graphs and for the different settings.

### 

### 

Figure . Prototype App on Phone

### Android Studio

We started our Android code with a REST API Android app base which we had created before during our Java Development for Android course. We then developed the interface of the app in parallel to this base model, following the Figma prototype as much as was possible with Android Studio tools.

Figma was a good tool for our code as it gave us Android code for each element, but it was also a trap for beginners like us. The code provided by Figma was difficult to piece together and adapt it to work as a whole and with the same result. It was timesaving to have this functionality but some of the elements did not work and some were just superfluous. This meant that we had to put a lot of effort to adapt this code into our code, and we quite quickly used less and less of the code it provided to us, using only the parts we knew would work.

For example, Android Studio does not have an easy, built-in option for creating circle images. We therefore had to use an external library called Circle Image View to create a circle image like in our Figma prototype.  
We also had issue when we superposed multiple layouts and boxes and some of our text disappeared.

Finally, we stopped using Figma code and used the information given to adapt our interface more correctly.

We deleted a lot of .xml files that Figma had given us which were unused, and we recreated our interface in a purer and more organized form, and this solved the layout and text bug.

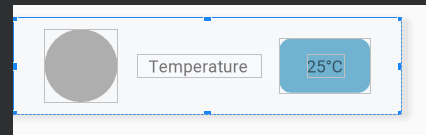
To manage the different information given by the sensors, we used a Card view layout that allowed us to have a template card with variables for the data name and value. The cards are then added to the interface with Java code and therefore it is very easy to add other sensors in the future, by just adding those sensors to a list and they will appear under the other cards.

Figure . Data Card Template



Figure . List of data cards added with the Java code

## Data collection and linking

### Rest Client

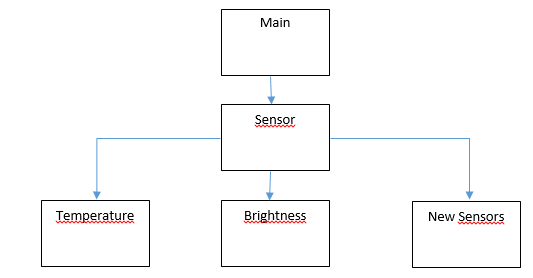
#### Android

#### Raspberry Pi

# Project perspective

## Planned additions

To improve the Python code and make it more organized and scalable for future additions of other sensors we decided to create the following class hierarchy:



## Challenges

# Conclusion