

# Degree Course Systems Engineering

Major: Infotronics

## Diploma 2014

*Yannick Baehler*

*Underwater rover and water  
quality monitoring*


- Professor  
François Corthay
- Expert  
Dr Rafat Ansari
- Submission date of the report  
30.10.2014





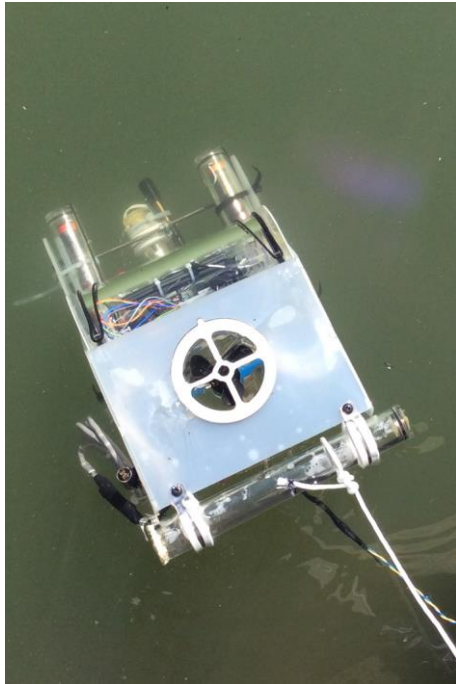
<input checked="" type="checkbox"/> FSI <input type="checkbox"/> FTV	Année académique / Studienjahr 2013/14	No TD / Nr. DA it/2014/59
Mandant / Auftraggeber <input type="checkbox"/> HES—SO Valais <input type="checkbox"/> Industrie <input checked="" type="checkbox"/> Etablissement partenaire Partnerinstitution	Etudiant / Student <b>Yannick Baehler</b>  Professeur / Dozent <b>François Corthay</b>	Lieu d'exécution / Ausführungsort <input type="checkbox"/> HES—SO Valais <input type="checkbox"/> Industrie <input checked="" type="checkbox"/> Etablissement partenaire Partnerinstitution
Travail confidentiel / vertrauliche Arbeit <input type="checkbox"/> oui / ja <sup>1</sup> <input checked="" type="checkbox"/> non / nein	Expert / Experte (données complètes) <b>Dr. Rafat Ansari</b>	

Titre / Titel  <b>Robot sous-marin</b>
Description / Beschreibung <p>Le projet OpenROV &lt;<a href="http://openrov.com/">http://openrov.com/</a>&gt; s'attache à développer un robot pour l'exploration sous-marine. Il est soutenu entre autres par la NASA.</p> <p>Le développement du système est hiérarchique : les capteurs et actionneurs sont pilotés par des circuits à microprocesseurs simples de type Arduino &lt;<a href="http://www.arduino.cc/">http://www.arduino.cc/</a>&gt;. Un processeur central pilote tous ces systèmes. Il permet de piloter le robot, de voir l'image fournie par la caméra et de récupérer l'information des capteurs, le tout à travers un navigateur web.</p> <p>Le but du projet est de développer le système de gestion des capteurs et de transmission des données à l'utilisateur via ces services web.</p> <p>Objectifs / Ziele</p> <ul style="list-style-type: none"> <li>– Equipement d'un sous-marin avec ses capteurs</li> <li>– Ecriture d'un code Arduino pour l'interfacage des capteurs</li> <li>– Ecriture d'une API Web pour récupérer les résultats de mesure.</li> </ul>

Signature ou visa / Unterschrift oder Visum  Responsable de l'orientation Leiter der Vertiefungsrichtung: .....  <sup>1</sup> Etudiant / Student : .....	Délais / Termine  Attribution du thème / Ausgabe des Auftrags: 12.05.2014 Remise du rapport / Abgabe des Schlussberichts: 30.10.2014 Défense orale / Mündliche Verfechtung: Semaine 46
---	---

<sup>1</sup> Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.  
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.





## Underwater rover and water quality monitoring

Graduate

Baehler Yannick

### Objectives

The objectives of this project were to build a system able to monitor the water quality in Lake Erie, which has severe problems of algae blooms, using an open source underwater rover.

### Methods | Experiences | Results

An open source underwater rover called OpenROV was first built from a kit and then modified by installing various sensors.

The temperature of the water can be taken using a thermistor and a small dedicated circuit, which allows to take very quick measurements of this parameter.

Electrical conductivity can be measured and gives information about the quantity of dissolved solids into water.

Light absorption through the water column can also be observed by the system, and gives information about the transparency of the water and the possible presence of chlorophyll.

All values gathered by the underwater rover are logged into a small embedded database that can be downloaded in order to be analyzed later on by a software developed in QT. This Software allows to plot various graphs of the data, visualize information about the measurements and export a summary of the measures taken to Google Earth.

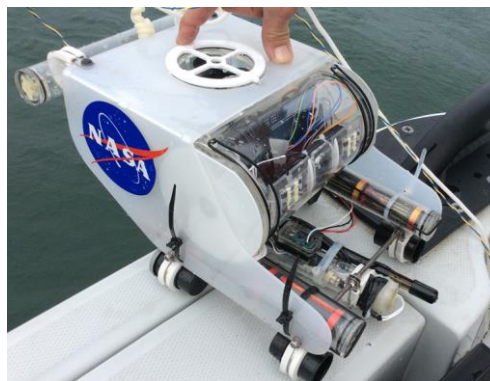
### Bachelor's Thesis | 2014 |

Degree course  
*Systems Engineering*

Field of application  
*Infotronics*

Supervising professor  
*Corthay François*  
*Francois.corthay@hevs.ch*

Partner  
*NASA Glenn Research center*  
*Dr. Rafat Ansari*



Final setup of the rover, equipped with various sensors able to monitor the water quality



Rover and other sensors used during measurements taken in Lake Erie



## CONTENTS

1	Glossary . . . . .	1
2	Acknowledgement . . . . .	1
3	Introduction. . . . .	2
3.1	Problem. . . . .	2
3.2	Objectives. . . . .	2
3.3	Report organisation . . . . .	3
4	OpenROV . . . . .	4
4.1	Introduction . . . . .	4
4.2	Rover presentation . . . . .	4
4.3	Hardware Architecture . . . . .	6
4.4	Software Architecture . . . . .	7
4.4.1	general architecture . . . . .	7
4.4.2	Plugins. . . . .	8
4.5	Serial communication . . . . .	9
4.5.1	OpenROV Controller side . . . . .	11
4.5.2	BeagleBone black side . . . . .	11
4.6	Version information . . . . .	13
4.7	IMU/depth sensor . . . . .	13
4.7.1	Presentation . . . . .	13
4.7.2	Software implementation . . . . .	13
4.7.3	Calibration . . . . .	13
5	Architecture . . . . .	14
5.1	Specifications . . . . .	14
5.2	Sensors adding . . . . .	14
5.3	Data Logging . . . . .	15
5.4	User Interactions . . . . .	15
5.5	Data Analysis . . . . .	16
5.6	General Architecture. . . . .	16
6	Conductivity sensor. . . . .	18
6.1	Introduction . . . . .	18
6.2	Architecture . . . . .	18
6.3	Sensor . . . . .	19
6.4	Connections. . . . .	20
6.5	Signal and communication . . . . .	21
6.6	Conductivity circuit. . . . .	21
6.7	Browser software . . . . .	22
6.8	BeagleBone Black software . . . . .	22
6.8.1	State machine . . . . .	22
6.8.2	Values forwarding . . . . .	25

6.9	OpenROV Controller board software . . . . .	27
6.9.1	Architecture . . . . .	27
6.9.2	Serial communication . . . . .	27
6.9.3	Sensor initialisation . . . . .	28
6.9.4	Communication with the sensor . . . . .	28
6.9.5	Temperature compensation . . . . .	28
6.10	Communication between circuits . . . . .	29
6.11	Calibration . . . . .	29
6.12	Tests . . . . .	30
6.12.1	Method . . . . .	30
6.12.2	Conductivity over range . . . . .	30
6.12.3	Precision test . . . . .	31
6.12.4	Field measurements . . . . .	32
6.13	Conclusion . . . . .	33
7	Temperature sensor . . . . .	34
7.1	Introduction . . . . .	34
7.1.1	Temperature . . . . .	34
7.1.2	Sensing solution . . . . .	34
7.2	Architecture . . . . .	34
7.3	Sensor . . . . .	35
7.4	Connections . . . . .	36
7.5	Thermistor Circuit . . . . .	36
7.5.1	Analyse . . . . .	36
7.5.2	Circuit . . . . .	37
7.5.3	Voltage regulator . . . . .	37
7.5.4	Divider . . . . .	38
7.5.5	Amplifier . . . . .	39
7.6	Signal filtering . . . . .	41
7.6.1	Design . . . . .	41
7.6.2	Simulation . . . . .	42
7.7	OpenROV Controller board software . . . . .	43
7.7.1	Analog-Digital conversion . . . . .	43
7.7.2	Noise filter implementation . . . . .	43
7.7.3	Voltage to temperature conversion . . . . .	43
7.8	BeagleBone Black and browser software . . . . .	45
7.9	Tests . . . . .	45
7.9.1	Accuracy . . . . .	45
7.9.2	Filter and precision . . . . .	46
7.9.3	Sensibility . . . . .	47
7.10	Calibration . . . . .	48
7.11	Conclusion . . . . .	48

8	RGB sensor . . . . .	49
8.1	Introduction . . . . .	49
8.1.1	Turbidity and transparency . . . . .	49
8.1.2	Visible light and algae detection . . . . .	49
8.1.3	Sensor's choice . . . . .	51
8.2	Architecture . . . . .	51
8.3	Sensor . . . . .	51
8.3.1	Measure of turbidity . . . . .	52
8.3.2	Detection of algae . . . . .	52
8.4	Connections . . . . .	53
8.5	Sensor positioning . . . . .	53
8.6	OpenROV Controller board software . . . . .	54
8.7	BeagleBone and Browser Code . . . . .	54
8.8	Calibration . . . . .	54
8.9	Tests . . . . .	55
8.10	Conclusion . . . . .	55
9	Database . . . . .	56
9.1	Database system choice . . . . .	56
9.2	Sqlite3 installation . . . . .	56
9.3	Database architecture . . . . .	56
10	Logging . . . . .	58
10.1	Architecture . . . . .	58
10.2	Data reception . . . . .	59
10.3	Database creation . . . . .	59
10.4	User view . . . . .	60
10.5	State Machine . . . . .	61
10.6	Logging . . . . .	62
10.7	Future development . . . . .	63
11	Rover modifications . . . . .	65
11.1	Introduction . . . . .	65
11.2	Main electronic tube modifications . . . . .	65
11.3	Buoyancy . . . . .	66
11.4	External electronic container . . . . .	67
11.4.1	Material and shape . . . . .	68
11.4.2	Mechanical design . . . . .	69
11.4.3	Endcaps assembly . . . . .	70
11.4.4	Fixation of the tube . . . . .	71
11.4.5	Final setup . . . . .	72

12	Data analysis program . . . . .	73
12.1	Introduction . . . . .	73
12.2	Architecture . . . . .	73
12.3	Mediator pattern implementation . . . . .	75
12.4	Interface . . . . .	75
12.5	Database . . . . .	76
12.5.1	Creation and utilisation . . . . .	76
12.5.2	Data access methods. . . . .	77
12.5.3	Future development of the class. . . . .	77
12.6	Measure module. . . . .	77
12.6.1	Architecture . . . . .	77
12.6.2	Overview class . . . . .	78
12.6.3	Analyse . . . . .	79
12.7	Parameters comparison . . . . .	80
12.7.1	Architecture . . . . .	80
12.7.2	Viewsettings class . . . . .	80
12.7.3	Analysecomp class . . . . .	81
12.8	Plots . . . . .	81
12.9	Google Earth exportation . . . . .	82
12.10	Conclusion . . . . .	83
13	Field tests . . . . .	84
13.1	Introduction . . . . .	84
13.2	Methodology . . . . .	84
13.3	Water parameters . . . . .	85
13.3.1	Temperature . . . . .	85
13.3.2	Conductivity . . . . .	87
13.3.3	Turbidity . . . . .	88
13.3.4	Water Colour . . . . .	90
14	Future development. . . . .	91
15	Conclusion . . . . .	91
16	References . . . . .	92
A	Directories organisation . . . . .	94
B	OpenROV structure diagram . . . . .	96
C	Conductivity commands. . . . .	98
D	Plugin Layout : Javascript . . . . .	100
D.1	file index.js (server side) . . . . .	100
D.2	file plugin.js (browser side) . . . . .	100
E	Plugin Layout : C++. . . . .	102
E.1	file plugin.h . . . . .	102
E.2	file plugin.cpp . . . . .	102



F	Octave script : Conductivity . . . . .	104
G	Octave script : Filter . . . . .	106
H	Data Analysis software class diagram . . . . .	108
I	Bill of material . . . . .	110
I.1	OpenROV . . . . .	110
I.2	Conductivity . . . . .	110
I.3	Temperature circuit . . . . .	110
I.4	Mecanical parts . . . . .	110
J	Database relational Schema . . . . .	112
K	Rover issues . . . . .	114
L	Endcap design: Acrylic sheet pieces. . . . .	118
M	Endcap design: Acrylic sheet Assembly . . . . .	124
N	Endcap design: Acrylic block . . . . .	126
O	OpenROV Controller Board schematics [13] . . . . .	128
P	BeagleBone Black schematics [14] . . . . .	138

## 1 GLOSSARY

In this report, it is assumed that the reader has basic knowledge in C++, Javascript and Node.js development. However, some essential words are described below in order to simplify the reading of this report.

Some essential information is given in the section 4 about the OpenROV structure. This couldn't however be done in detail in this report. Some words necessary to the understanding of the ROV are also described below.

`Cockpit interface`: Refers to the web page allowing to control the ROV. This webpage can be accessed by connecting to the rover by its IP address from a Google Chrome browser.

`Cloud9 interface`: Refers to the web page allowing to develop on the ROV. This webpage can be accessed by the same way than for the cockpit interface.

`Global event loop`: Loop that calls all the scripts in the Javascript architecture.

`Module (Node.js)` : A module in Node.js is a small library that can be downloaded to provide extra high level functionalities

## 2 ACKNOWLEDGEMENT

I'd like to gratefully acknowledge the assistance of a few people that gave a precious help along this projet.

Firstly, I'd like to express a special thanks to `Dr. Rafat Ansari`, who made this project possible and provided an appreciated help.

And to `François Corthay`, for his advices along the project.

And to `James King`, for his logistic help and advices.

And to `Bob Paulin` and `Dan Gedeon`, for their precious help and assistance during the project.

### 3 INTRODUCTION

#### 3.1 Problem

This project was realized in the frame of the study of algae blooms in lake Erie, which is a big concern for the water quality. Those blooms appears for a few week during the summer months and cause notable troubles against the wellness of the population and local wildlife. Understanding how those blooms develops and how to predict them are for this reason, a very important matter.



Figure 1: Satellite photography of lake Erie during algae blooms

Various techniques can be used in order to monitor the lake, each one having its pros and cons and all of them are needed in order to have a full scope monitoring. Satellites can be used to assess the water quality and can be very precise to observe the presence of certain elements into water, such as algae or sediments, by using spectroscopy. However, the information gathered is not real time and are also not always possible. The presence of clouds can make it difficult or the satellite can be in an unfavourable position. Aircraft or unmanned aerial vehicles can be used for spectroscopy as well and are sometimes able to gather those data when satellites can't. Aerial vehicles are also more accessible than satellites which make their use essential. Remote sensing can however only give data concerning some of the parameters affecting the water quality. Data that can be collected this way also concerns mostly surface phenomenon and thus, can't observe what is appening beneath. This makes in situ monitoring essential to have a full understanding of the

state of the lake.

Various approaches can be used for field monitoring. Collecting samples in the lake stays the only way that allows a full analysis of what is inside of the water. It is however very demanding in time and money. Samples must be collected and analysed in a lab, which can take an enormous amount of time.

An other approach, which is the main interest in this project, is the analysis of water using sensors. Using sensors can't give an analysis as complete as samples analysis for a given location, but it is very effective to have a general view of a specific area of the lake and detect changes that might occur. Even appearing small changes in a water quality parameter can have an impact on the lake ecosystem and detecting them can reveal what is going on inside the water and give an insight of what future consequences it might cause. Constant monitoring of specific places throughout the year can then serve as an early alert of algae bloom and potentially predict their intensity.

#### 3.2 Objectives

This project was realised to serve as a proof of concept that in situ monitoring can be done both effectively and inexpensively. It consisted in setting up a small robotic sensor platform, able to monitor a small portion of the lake. This robot was chosen to be a small open-source underwater rover, called openROV, that had to be modified to be equipped with water quality sensors. A few sensors were added during this project: temperature, conductivity, colour and light. However, some other might be added in the future.

For this project, an important objective was to keep a price as low as possible without compromising the viability for an effective monitoring of the water quality. For this reason, expensive sensors such as turbidity meter or fluorometer, which would give precious information in the study of algae, must be avoided and replaced by cheaper sensor able to give information about the same parameters. To this end, a light meter allows to monitor the transparency of the water and thus, give information similar to what a turbidity meter would give for a much lower price. A colour meter can even detect the presence of chlorophyll into water and replace a chlorophyll detector for also a much cheaper price, despite the fact it will never be as precise. Conductivity, without giving precise information of the dissolved solids into water can give a good estimation and detect important changes. Temperature can be easily measured and important changes can have devastating effect on the wildlife.

Another objective of this project was to facilitate the addition of sensors without having to modify what is already in place. By chance, the OpenROV structure is already organized in a way that helps in this matter, but this had to be considered for the logging and visualization of data.

### 3.3 Report organisation

This report is organised in the following manner :

In the section 4, the OpenROV rover, used for this project is explained. Its software architecture and its capabilities must be known in order to design a proper architecture for the water monitoring capabilities. The Architecture developed is then described in the section 5.

The sensors that were installed during this project and the water quality parameters they measure are then explained one by one in their corresponding section. What led to their choice and how their specific software was designed is described.

Once the sensors were installed, data obtained from them needed to be logged in an organized manner, in order to be analysed later on. This was made by logging the data into a small database, which will be explained in section 9. A dedicated software, described in section 10, handles the gathering of the sensor's data and their logging into this database.

The addition of those elements necessitated some modifications of the original mechanical design. The main problem was to add the extra electronic needed by the sensors to the OpenROV, in which few remaining places were available. Another thing to consider was the correction of the buoyancy as sensors were added. Those modifications are explained in section 11.

The last development for this project was to create a program able to visualize the gathered data and analyse them, by plotting various graphs and displaying some information. This program will be explained in section 12.

At last, tests were performed in various locations into Lake Erie. Data were gathered and briefly analysed in order to verify that they could be used for water monitoring.

## 4 OPENROV



Figure 2: OpenROV [12]

### 4.1 Introduction

The first step for this project was to build the rover, called OpenROV, which is an Open source underwater rover able to reach a depth of 100 meters. It serves as a basis on which the project can be constructed without having to develop the mobility aspect. This allows to focus on water monitoring instead of robotics.

Once built, an important step was to understand the way the openROV works. The purpose of this chapter is to give an overview of the OpenROV architecture, which is important to clearly have in mind in order to be able to understand the choices that were made throughout the project.

In this chapter, information about the OpenROV and its architecture will be given. An emphasis will be made on two important elements for further development: The communication between two electronic boards within the rover, and how additional software can be added to the existing structure. A plugin system was created by the OpenROV team in order to facilitate the addition of software, but few documentation about it was available, which is why the important principles need to be explained here. Moreover, the OpenROV is in constant development and what was used during this project might change in future versions.

### 4.2 Rover presentation

The OpenROV, visible in the figure 2 is structured as follow :

Two tubes, positioned at each side of the ROV contain the batteries that power the system. Each tube contains 3 batteries of 4 volts, connected in series, that provide 12 volts for each tube. Those two tubes are connected to the OpenROV electronic to power the whole system.

All the electronic is contained inside the main electronic tube, which include various circuits and a camera. This container is closed by two caps on each sides that can both be opened. All wires going to the outside of the tube are sealed into those caps.

The rover is able to move by means of 3 propellers : Two propellers, placed at the back of the ROV, allows it to move forward, backward and to turn. An other propeller, placed at the top of the ROV, allows it to move vertically into water.

There are two ways by which an underwater vehicle can move vertically into water: The vehicle can use an active buoyancy control system, using generally hydraulic pumps to change its weight underwater, or it can be neutrally buoyant, using some form of propulsion to change its elevation. The OpenROV is neutrally buoyant since it uses its vertical propeller to move vertically. It is the cheapest option, but it requires to have the rover's weight adapted for each change in set-up, such as the addition of sensors.

The figure 3 below shows where the elements described above are placed onto the rover :

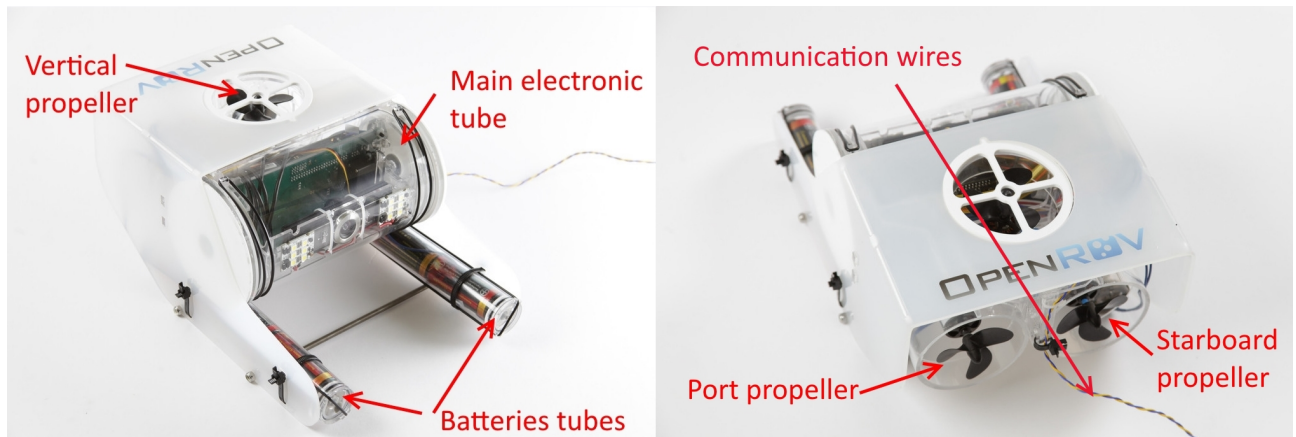


Figure 3: OpenROV: front and back views [12]

The rover can be used with any computer having a Google Chrome browser installed. The computer is connected to a little box that allow to use only two wires, that serve as tethers, for the ethernet communication. This is shown in the figure 4 below :

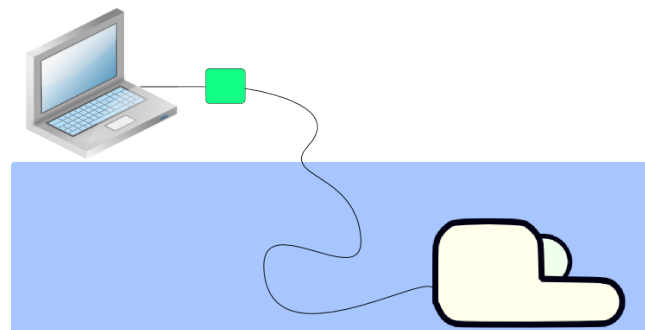


Figure 4: Rover driven by the computer

### 4.3 Hardware Architecture

A first block diagram can be established showing a global view of the system and can be seen in figure 5 .

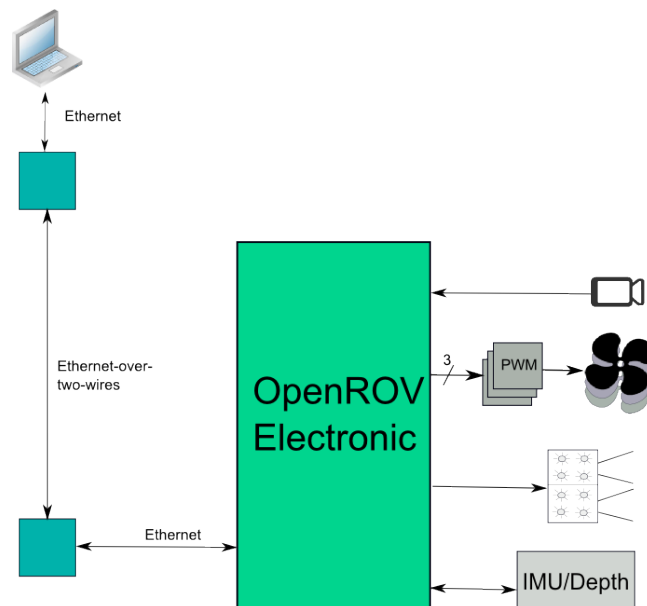


Figure 5: Block diagram of the OpenROV

The computer and the rover are connected with a twisted pair through the water. The communication between them is done using the Ethernet protocol but the signal must be converted to pass through a single twisted pair, at both ends.

The electronic in the rover is connected to various input and output as it can be seen in the figure 5. Not all connected devices are shown in this picture but only the essential ones. It can be seen that a sensor is already installed, the IMU/Depth Sensor, which will be described more precisely in the section 4.7.

The block diagram in figure 5 can be upgraded to show more precisely the various elements composing the electronic of the system:

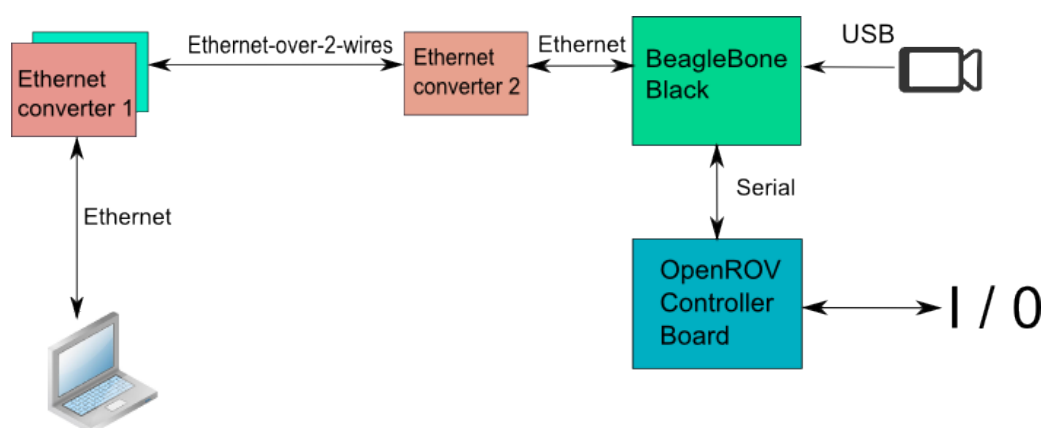


Figure 6: Block diagram of the OpenROV's electronic

Various elements can be distinguished :

The OpenROV Controller Board v2.6 has the purpose to make the connection with the external devices. All sensors and actuators are connected to this board such as the engines for the propellers and the servomotor that allows to move the camera up and down. This board embedded an ATMEGA2650-16AU micro-controller. The schematics of the OpenROV Controller Board can be found in annexe O.



This controller board is connected to a `BeagleBone Black` and can communicate with it through a serial connexion. The `BeagleBone` has most of the computing power for the ROV and allow the connexion between the ROV and the PC that controls it. This board embedded an `AM335x 1GHz ARM Cortex-A8` micro-controller. Unlike the other devices, the Camera is connected to this board instead of the `OpenROV Controller Board`. The schematics of the `BeagleBone` can be found in annexe P

The `BeagleBone Black` is connected by a twisted pairs cable to a circuit, called `Ethernet converter 2` in the figure 6 (which is a part of a `Tenda P200 Powerline adapter`). This circuit converts the communication signal to a single twisted pair.

At the surface, this 2 wires communication is converted back to a standard twisted pair cable that can be connected to the computer from which the ROV is driven. This conversion is made by two circuit, called `Ethernet converter 1` in the figure 6, which are a `Tenda P200 circuit` and a `OpenROV Topside Interface Board v2.6`.

The system can be expressed more precisely in form of an UML structure diagram, that can be seen in annexe B.

## 4.4 Software Architecture

### 4.4.1 general architecture

The software is dispatched between the different elements as shown in figure 7:

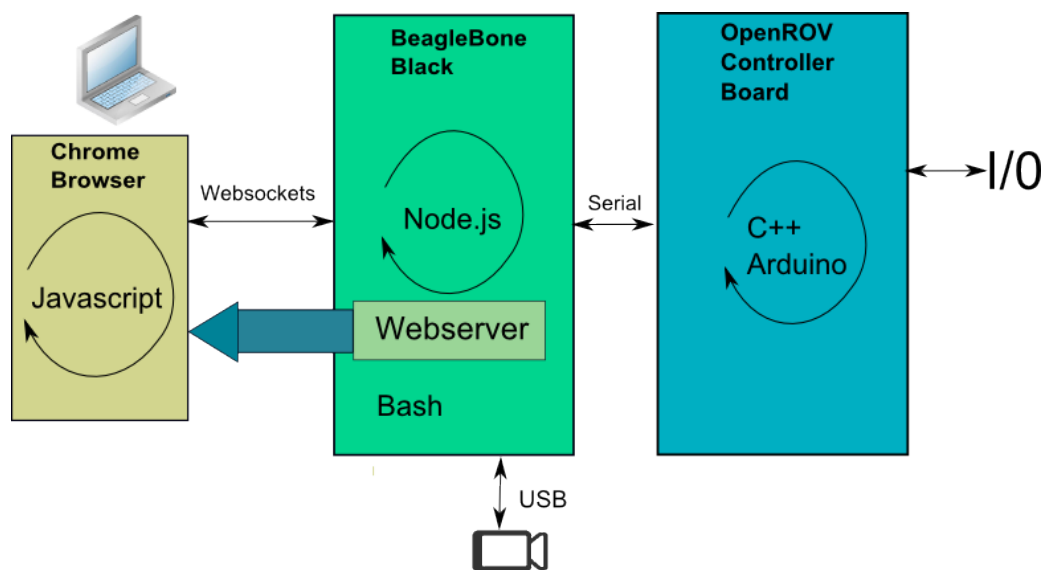


Figure 7: OpenROV software global architecture

The software on the `OpenROV Controller Board` is a `C++` code using an `Arduino` library. This code controls the sensors and actuators at a low level.

The software on the `BeagleBone Black` is a `Node.js` code and is charged to control the ROV. It also implements the web server that provides the web pages allowing to control the rover.

Web pages, accessed through a `Chrome` browser, allow to control the ROV. Those pages are written using `Javascript` and can communicate with the `Node.js` code on the rover through web-sockets.

Although the compiled code is dispatched between the `BeagleBone Black` and the `OpenROV Controller Board`, as shown in figure 7, all the source files are located on the `BeagleBone Black` and bash scripts on it allow to compile the `C++` files and send them into the `OpenROV Controller board`. The organisation of all the scripts and source code files into the `BeagleBone Black` can be seen in the annexe A.

The software is working through 3 loops: the `JavaScript` global event loop for the browser, the `JavaScript` global event loop for the `BeagleBone Black` and the micro-controller loop for the `OpenROV Controller Board`.



The different elements composing the OpenROV software are all called from those loops, as it can be seen in the figure 8 below:

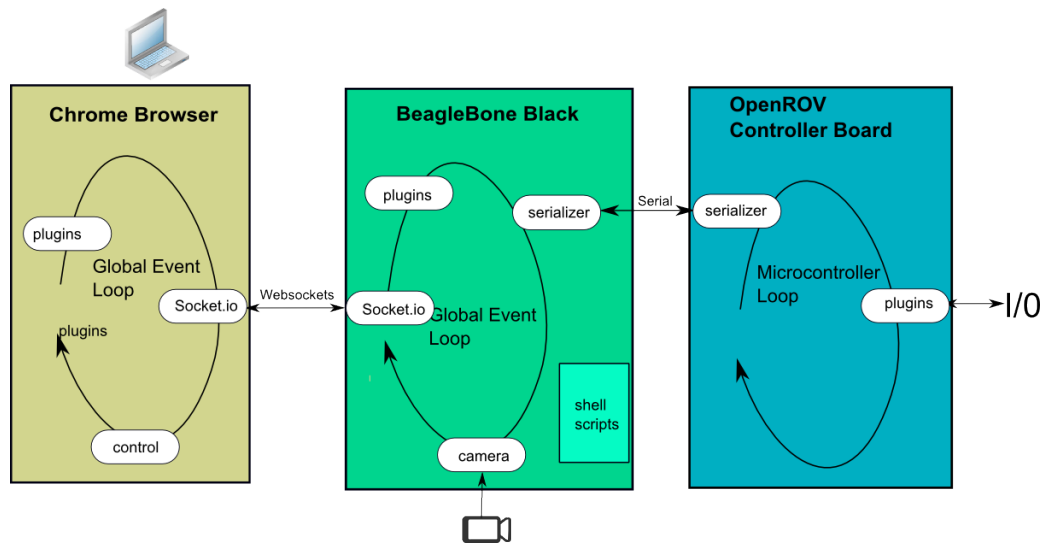


Figure 8: OpenROV software architecture

#### 4.4.2 Plugins

Among all the files that form the OpenROV software, new feature for the OpenROV can be implemented as plugins. Almost all the code that needs to be written to add sensors to the OpenROV can be added as such.

Almost all devices on the rover are driven using the sensor structure, which can be illustrated as in the figure 9 below. It can be seen in that figure that the plugins follow exactly the general software architecture and new features can be added for all devices.

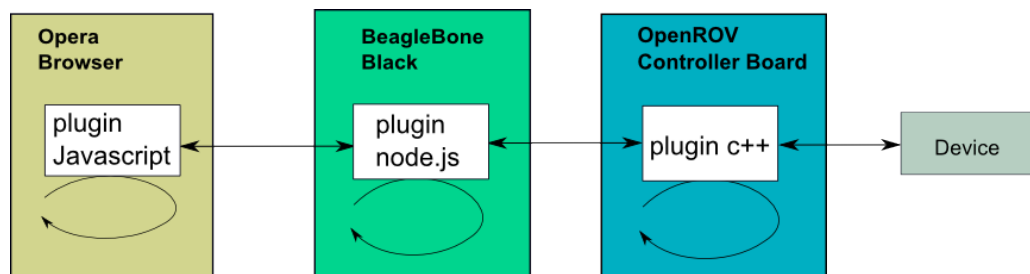


Figure 9: Plugins

For the OpenROV Controller Board, plugins can be written in C++, using an Arduino library. The files are present on the BeagleBone in the folder `/opt/openrov/arduino/OpenROV` but they are executed on The OpenROV Controller Board. They need to be compiled and uploaded on the right board by using the script `/opt/openrov/linux/arduino/firmware-installfromsources.sh`.

Writing code for this board allow to handle a low level access to the hardware connected to it. Each plugin for this board must be derived from the class `Device` and is called from the micro-controller loop defined in the file `OpenROV.ino`. All plugins developed must have a method `device_loop(Command)` that is called during the main loop and a method `device_setup()` called at initialisation.

The procedure to create a new plugin file is the following :

- ◆ Create the new C++ files (header and code) in the folder `/opt/openrov/arduino/OpenROV`. Templates for those files are available in annexe E.

- ◆ The OpenROV.ino that initiate the main loop needs to be modified by adding the following code :

```

1
2  \#if (HAS\_PLUGIN\_NAME)           //defined in ./Aconfig.h
3  \#include "PluginName.h"          //inclusion of the header
4  PluginClass plugin;               //Initialisation of the object corresponding to the created class
5  \#endif

```

- ◆ The file Aconfig.h must be modified to indicate that the new plugin is available

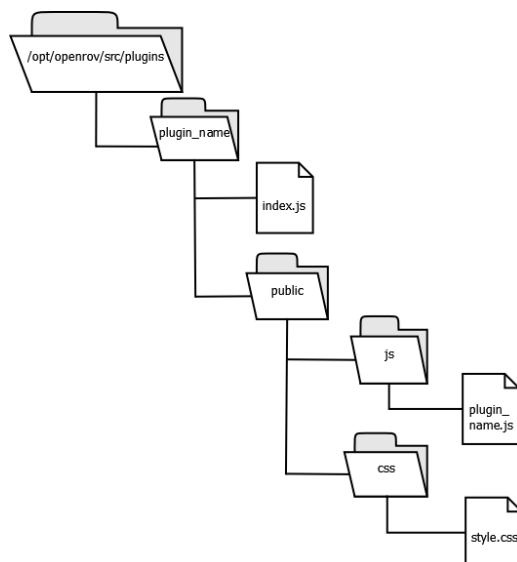
```

1  \#define HAS\_PLUGIN\_NAME (1)

```

This procedure was obtained from the file /opt/openrov/arduino/OpenROV/Readme.md on the BeagleBone .

The development of an additional plugin for the BeagleBone Black and the browser can be done by creating a new folder in /opt/openrov/src/plugin. This folder can be named with the name of the project and have a specific structure, shown in the figure 10.



The file `index.js` regroups what is executed in the Node.js event loop. The content of this script is not displayed in the browser and contains the server side code.

The file `public/js/plugin_name.js` contains what is executed in the javascript event loop, executed in the browser. It can communicate with the server side code through web sockets. It can also contain the HTML, which could also be in an external file if it is too heavy.

CSS files can be added in the folder `public/css` for specific layout option for the plugin.

Figure 10: File organisation of the plugins

Those files are automatically added into the global event loops when the openROV cockpit program starts. No code needs to be written nor file be modified in order to make the plugin work. The `index.js` and `plugin_name.js` have however to follow a specific layout, that is given in annexe D.

## 4.5 Serial communication

The OpenROV Controller board and the BeagleBone black communicate through a serial communication. This section will explain how it is made for each circuit and how the commands need to be formatted to be properly parsed.

Messages can be sent from both way and must use a precise message formatting, which depends on the message's direction. The formats to use for each direction can be seen in the figure 11 below:

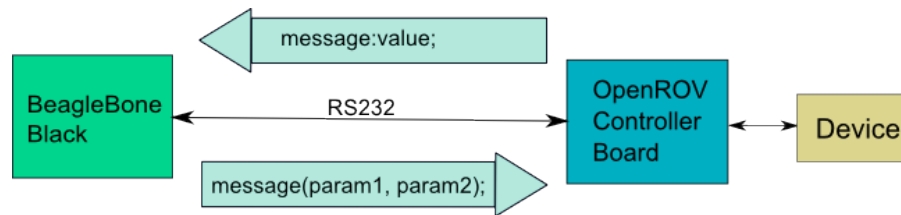


Figure 11: Serial communication : messages format

It is important to use always the same format to avoid conflict between the added messages and the rest of the OpenROV program. It is also very important to check that every message name is unique within the entire program, otherwise, conflicts could occur.

The serial communication between the OpenROVController board and the BeagleBone black is done with the following classes :

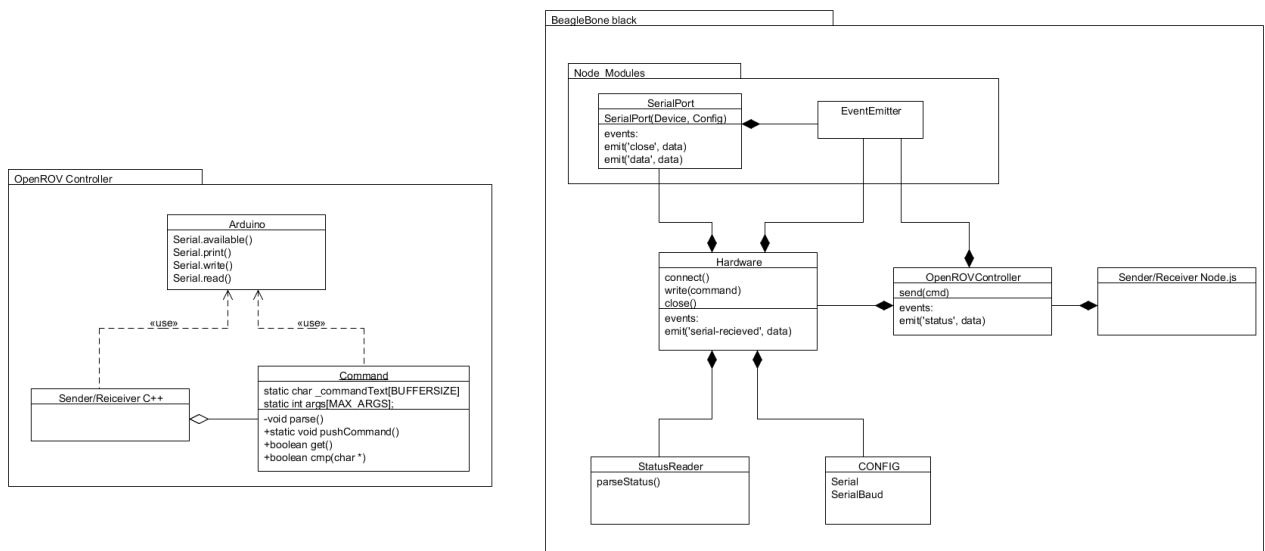


Figure 12: Serial communication : UML Class diagram

The serial communication is made between the Node.js `SerialPort` module and the `Serial` object of the `Arduino` library, each handling the serialization for its board.

The classes called `sender/receiver` in the figure above are the plugins corresponding to a device.

#### 4.5.1 OpenROV Controller side

A plugin on the `OpenROV Controller` board receives and sends commands as shown in the figure 13 below :

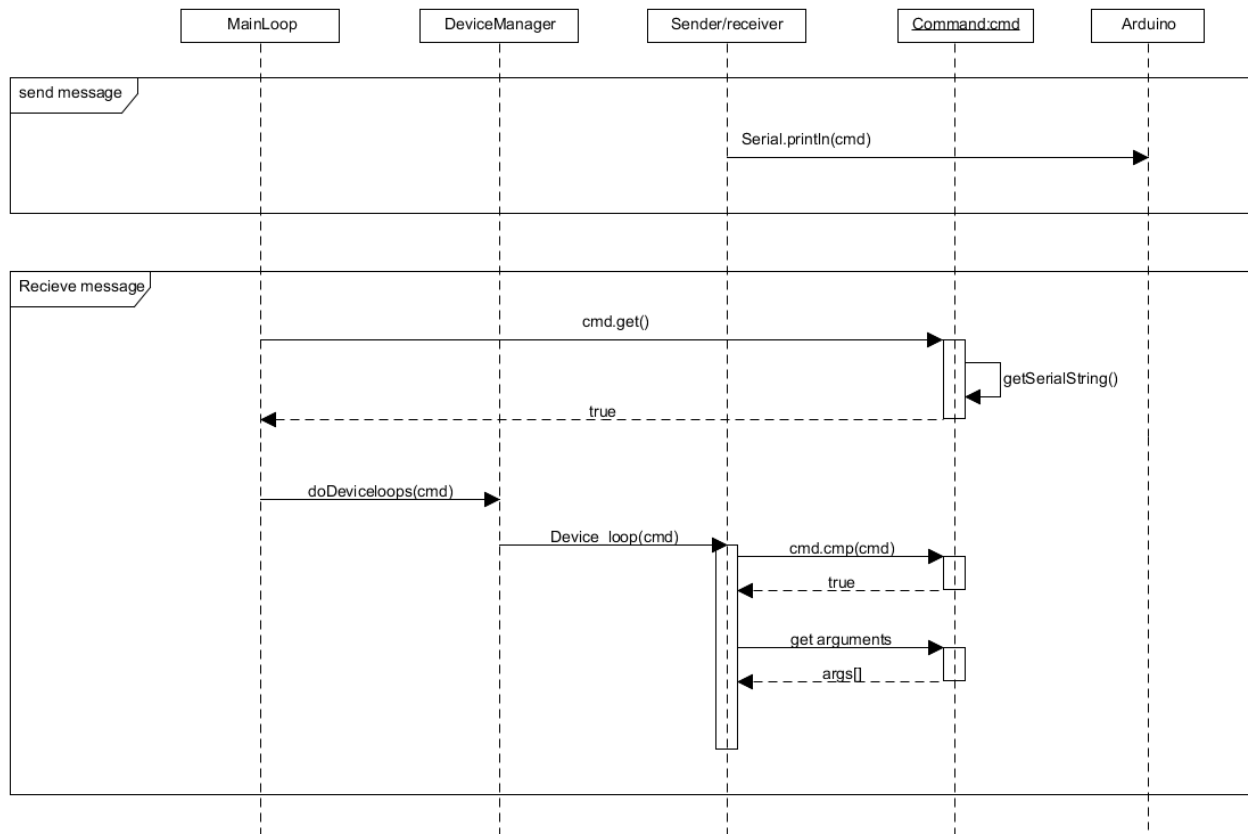


Figure 13: Serial communication: OpenROV Controller board side

The serialization on the `OpenROV Controller` board side is made through the `Arduino` library. The plugin(C++ file) wishing to send messages to the `BeagleBone` can directly use the `Serial` object of this library to send a message.

However, the reception is a bit different since the serial stream needs to be parsed and then interpreted by the concerned device's files.

A `Command` object is instantiated into `OpenROV.ino` file in which the main loop is written. At each iteration of the main loop, the method `get()` of the `Command` object is called, which will parse the serial stream and put one of the commands as a current command. Later in the loop, The `Command` object is sent to all plugins through their `device_loop` method.

Each plugin can verify if they need the current message through the method `cmp(char*)` of the `Command` object. If the plugin need to handle this message, it can get the arguments of this command by checking the attribute `args[]` of the same object.

It must be taken into consideration than once the main loop has finished an iteration, the current message will be discarded.

#### 4.5.2 BeagleBone black side

On the `BeagleBone Black` side, the communication is made through the `Node.js` module `SerialPort` that handles the serialisation. This module is required by the `Hardware.js` file, by which handles most of the communication.

The message exchanges on the BeagleBone Black can be seen in the figure 14 below for both sending and receiving a message :

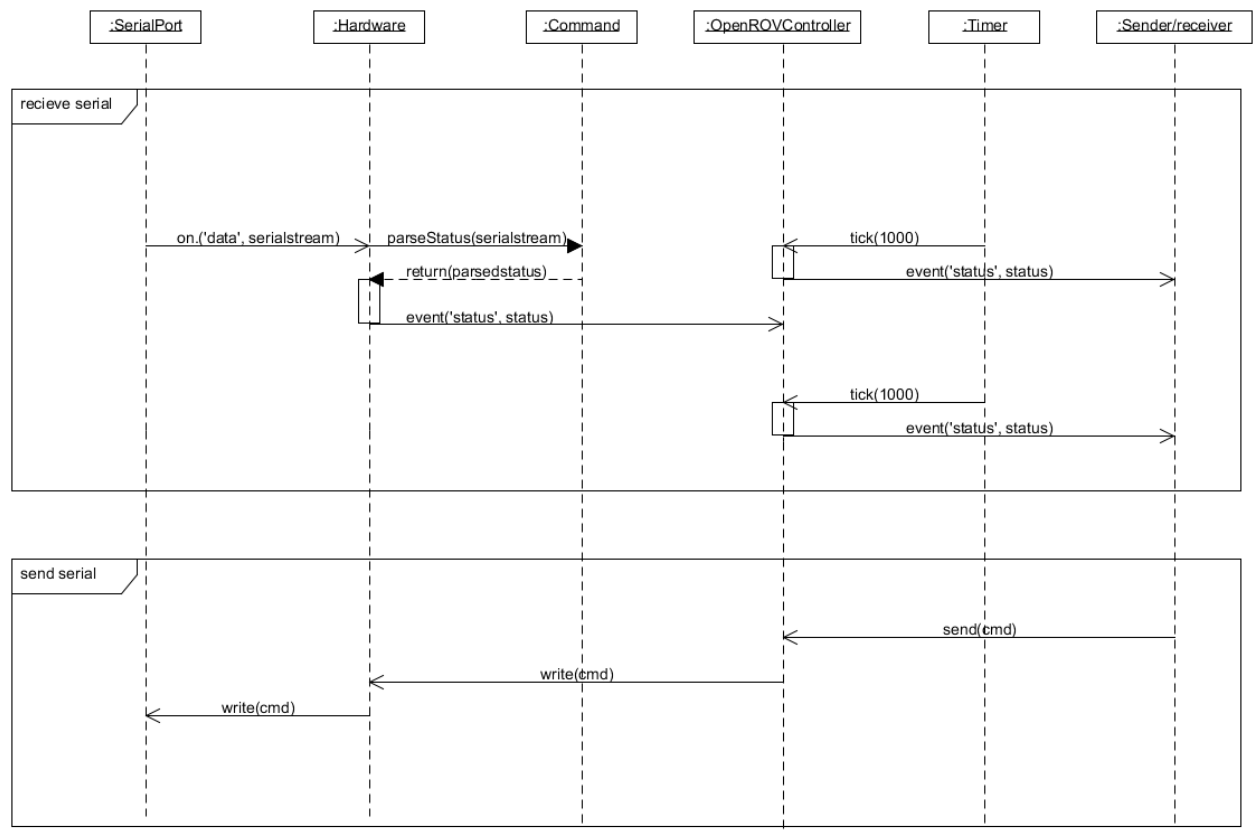


Figure 14: Serial communication: BeagleBone black side

When sending a message to the OpenROV Controller Board, the Hardware.js file uses the StatusReader.js file to parse what is received from the Controller Board. It then emit an event containing a parsed status composed of values indexed by names that is received by the OpenROVController.js file. This last file is then charged to send this status every second as a global Event, which is available for all plugins.

It is necessary to pay attention to this point when developing a plugin that communicate through the serial port. If the status changes during the 1 second interval, the previous message is discarded. Moreover, the state of the status is never erased. This means that messages such as events that needs to be handled as fast as possible and/or only once must be identified and emitted separately from the status event. In those cases a modification of the OpenROVController.js file is necessary. This file can be found in /opt/openrov/src/libs on the BeagleBone Black.

Sending a message to the Controller Board is made by using the function `send(cmd)` of `OpenROVController.js` which will relay the message to `Hardware.js` that will sent it through the `SerialPort` module.

## 4.6 Version information

The OpenROV version used during this project is the v2.6, bought in May 2014

A Software image was pre-installed on the BeagleBone Black, but can be downloaded on the OpenROV GitHub (<https://github.com/OpenROV/openrov-image>). The software that was installed for this project is the version 2.5

The OpenROV Software (in the folder /opt/openrov in the BeagleBone Black) was updated last in June 2014.

## 4.7 IMU/depth sensor

### 4.7.1 Presentation

An extra sensor can be bought with the OpenROV and was added to it for this project. This sensor is in fact a combination of four sensors : an accelerometer, a magnetometer, a pressure sensor and a temperature sensor. Those were assembled in a circuit by the openROV team and can be connected to the I2C bus. Those sensors allow to know the positioning of the ROV and give the driver useful information. The depth and the temperature are as well important for the water quality monitoring.

### 4.7.2 Software implementation

Software necessary to get the values from the sensor is already written and ready to use. This code includes the recuperation of data, their processing, the communication between the OpenROV Controller Board and the BeagleBone, a graphical view visible directly on the camera image and a calibration interface.

Some file need however to be modified in order to run the corresponding code.

The AConfig.h file needs to be modified to reflect the new hardware. The following lines need to be updated to have (1) :

```
1 #define HAS_MS5803_14BA (1)
2 #define HAS_MPU9150 (1)
```

### 4.7.3 Calibration

A calibration interface is available in the Cockpit interface under the Diagnostic tab and allows to set the depth at 0 for the current pressure value, change the water type (in salt water the pressure will be greater and the depth derived from it will change). It is also possible to calibrate the magnetometer.

No changes were made to the code.

## 5 ARCHITECTURE

### 5.1 Specifications

Once the structure of the OpenROV software understood, an architecture could be elaborated.

The features asked for this project were the following :

- ◆ Sensors must be added to the rover in order to monitor the water quality in lake Erie
- ◆ A program allowing to visualise the data taken by the rover must be created

With those features, a few important points to take into consideration were fixed :

- ◆ The system must be low cost
- ◆ Additional sensors should be added without important modifications to the existing features
- ◆ The system must be user friendly

No specifications were given about the sensor to install, they had to be chosen by considering the low cost aspect asked and a viable water quality monitoring. The sensor that were chosen are :

- ◆ A conductivity sensor
- ◆ A temperature sensor
- ◆ An RGB/light sensor

The choice of those sensors will be explained in their corresponding section of this report.

### 5.2 Sensors adding

Additional sensors must be added to the OpenROV in the most cost efficient way. In order to do so, the best way is to connect the sensors directly to the OpenROV Controller Board. This board was made at this effect and has plenty of I/O available.

Another solution could have been to use an other board at this effect, but the place being limited into the main electronic tube, an extra waterproof container would have also been needed. The addition of both an extra container and an electronic board would have increased the price of the system which needed to be avoided.

Moreover, if a circuit had to be added, another problem would emerge, which is the communication with the surface. In this case passing by the OpenROV electronics in order to use the existing communication medium would be the easiest way, but has the same drawback to require the use the OpenROV structure. Adding extra wires and developing an alternate communication would take a tremendous amount of time and was out of question.

For all those reasons, using the existing architecture and connect the sensors directly to the OpenROV Controller Board was chosen. However, adding an additional circuit, even if this has not been implemented in the current setup could be done in the future in order to increase the amount of sensors if no additional I/O are available.

This choice forces to use the OpenROV software architecture and use the implemented serial communication and code structure, as it is described in the section 4

The OpenROV Controller board can communicate at a low level with the additional sensors by the C++/Arduino plugins and send the values of the sensors to the BeagleBone black. The BeagleBone can then implement various needed functionalities for each sensor and handle the communication with the browser, which can display data and provide interfaces. All those steps can be done without modifying the existing code, by simply adding new plugins.

This gives the architecture shown in the picture 15 below:

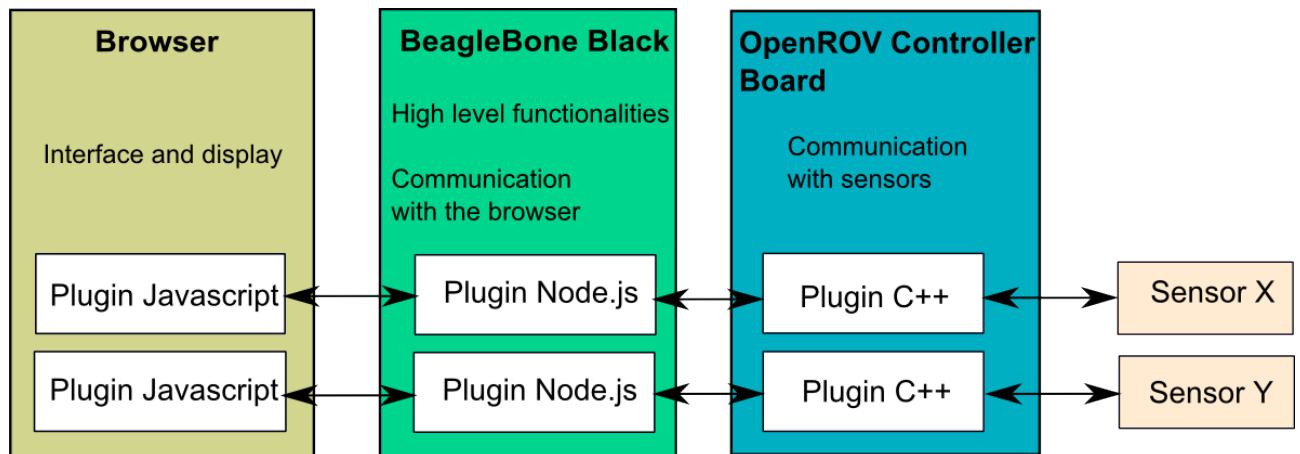


Figure 15: Sensors communication architecture

### 5.3 Data Logging

Measures taken by the sensors need to be logged for later analysis. This logging system should be able to handle additional sensors without any structure modifications. By doing so, an analysis software can be written independently from the number of sensors and facilitate the extension of the system.

Whatever the format in which the measures are registered, a location for this file must be chosen. It can either be stored in the rover or in the computer driving it. Storing the data in the rover is the best solution as it makes the logging system independent from the communication with the computer, which decreases the complexity and avoid problems that could occur during the transmission of data. It also keeps the OpenROV philosophy to be compatible with any computers without any software to install other than a browser.

The only possible drawback might be the available storage place on the BeagleBone Black, but the BeagleBone Black has an available SD card slot that is not used which makes this problem insignificant if a light weight format is used for the data.

The format chosen to store those data is a SQLite database. This choice will be explained in the section 9.

Another point to take into consideration is how the data logging can be done in the OpenROV architecture. It was chosen to implement a plugin that gather all the values available in order to log them, those values being sent by each sensor's Node.js plugin through global events. This plugin should only get all data available but not modify them in order to keep a clear organisation.

The organisation of the logging in the BeagleBone Black software can be represented as in the picture below:

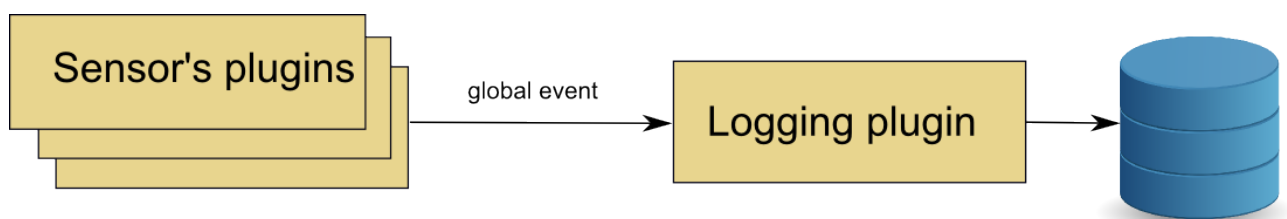


Figure 16: Data logging

### 5.4 User Interactions

The user needs to be able to interact with the different sensors. The OpenROV plugins on the BeagleBone black can be developed for both the server and the browser side, which allow to implement those interaction in the same plugin than the sensor concerned. Interactions needed for the logging plugin can also be implemented by this mean.



## 5.5 Data Analysis

Different approaches were taken into consideration for the analysis of the data :

- ◆ A module included into the rover software, which could be accessible from the web interface
- ◆ A software developed for the computer, that could communicate with the rover through web-sockets to retrieve the data
- ◆ A software developed for the computer, that could use a file previously downloaded containing the data

The approach of an independent software that uses its own file was chosen. The main advantage of doing so is the separation between the analysis software and the rover. The ROV is not needed and data can be used simultaneously by different people at different locations, which is not the case for the other solutions. Moreover, if the user decide to write new measurements into a new file. It is easy to simply download the current file and archive it before creating a new one. Both file can then be analysed with the software at any time.

This solution is illustrated in the figure 17 below:

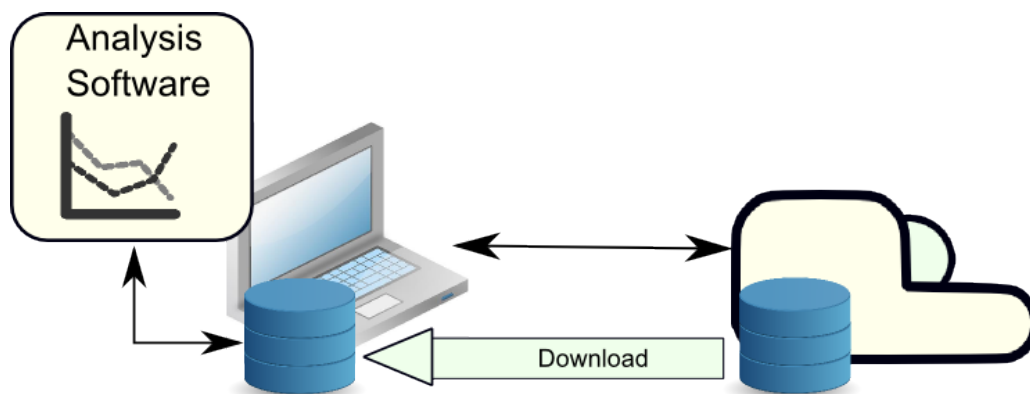


Figure 17: Database and Analysis program

A drawback for this kind of program is that an analysis is not possible during a measurement. This problem was discussed and it has been decided that such a functionality doesn't need to be implemented.

## 5.6 General Architecture

Considering the elements discussed above, the architecture for logging and analysis becomes as shown in the figure 18 below. All Sensors that were added during this project are visible on this figure, but the choices that lead to those sensors will be explained in their corresponding section of this report.

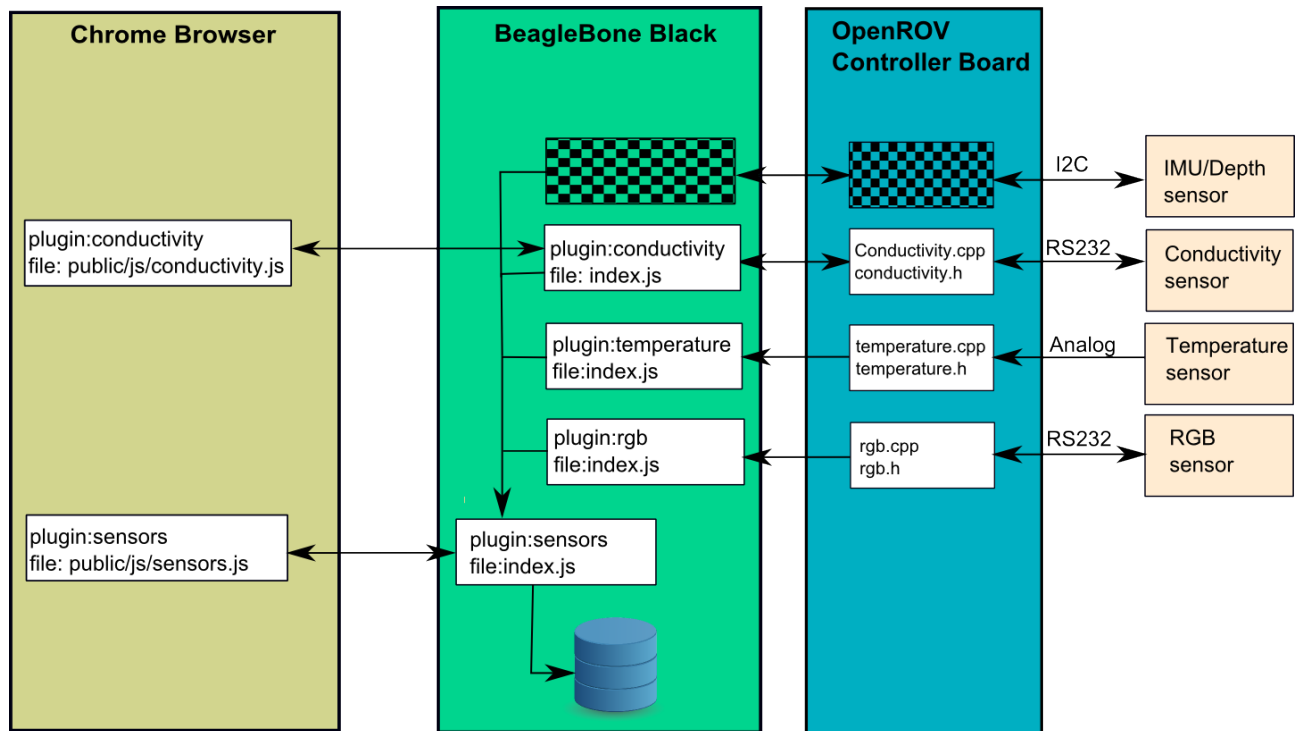


Figure 18: Sensors communication architecture

The files that need to be created all follow the OpenROV plugin architecture. Therefore, the location and name of those files can already be defined. The plugins for the IMU/depth sensor are checked in the figure above because they were not modified during this project. The only thing that was done for this sensor is to get the value of the depth and temperature directly from the status event send in the global event loop by the OpenROVController.js script, which is entirely implemented in the logging plugin and doesn't require any modification of the IMU/Depth's plugins.

## 6 CONDUCTIVITY SENSOR

### 6.1 Introduction

Electrical conductivity is an extremely useful water quality parameter, very commonly measured. It can be used as a base for salinity or TDS (total dissolved solids) estimations as they are strongly related [9]. Electrical conductivity is also especially useful as an early indicator of changes in water quality, since most bodies of water maintain a fairly constant conductivity that can be used as a baseline of comparison to future measurements [9].

It is however important to keep in mind that the salinity and TDS values derived from the conductivity can't reflect the exact composition of the water, since such precise measurement are only possible by collecting samples and analysing them in a lab. Those values are derived from simple coefficient determined by the composition of the water. From this, it can be understood that salinity for example makes sense only in sea water where the amount of NaCl present into the water is considerable comparing to other solids and can be for this reason estimated from a simple coefficient. Salinity is then not especially interesting for this project, TDS however, is a very useful measurement in lake water that is derived in a similar way than salinity using simply a different coefficient.

A change in the water composition will either increase or decrease its conductivity, which makes the conductivity measurement useful as an early indicator. The presence of new Ionized dissolved such as phosphate (which is a known factor of algae bloom in lake Erie) will increase the conductivity of the lake [2]. At contrary, a large presence of not ionized dissolved solids such as organic compounds in water can cause the lake to have a low conductivity since they don't conduct electricity very well [2].

### 6.2 Architecture

The general architecture for the conductivity measurement is as shown in the figure 19 below:

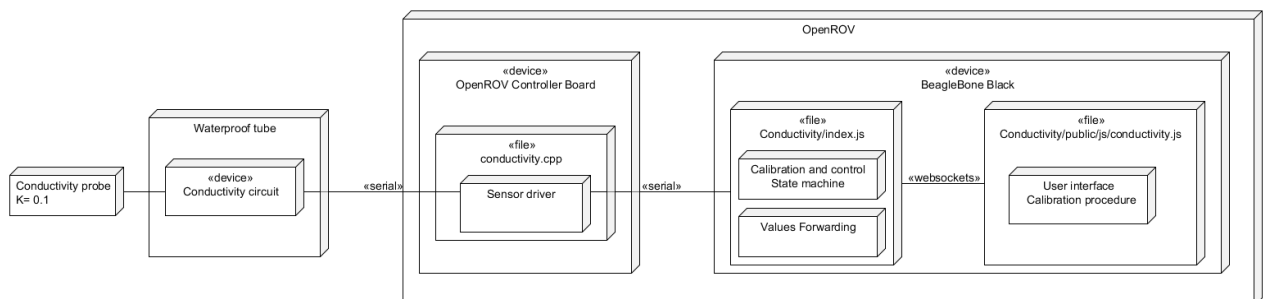


Figure 19: Architecture of the conductivity measurement

The Conductivity probe is connected to a small circuit that translates the probe signal into a conductivity value. This circuit is located into a small waterproof container and is then linked to the OpenROV Controller Board into the main electronic tube.

The software part of this architecture is implemented using the OpenROV plugin system and is shown in the figure 20 below :

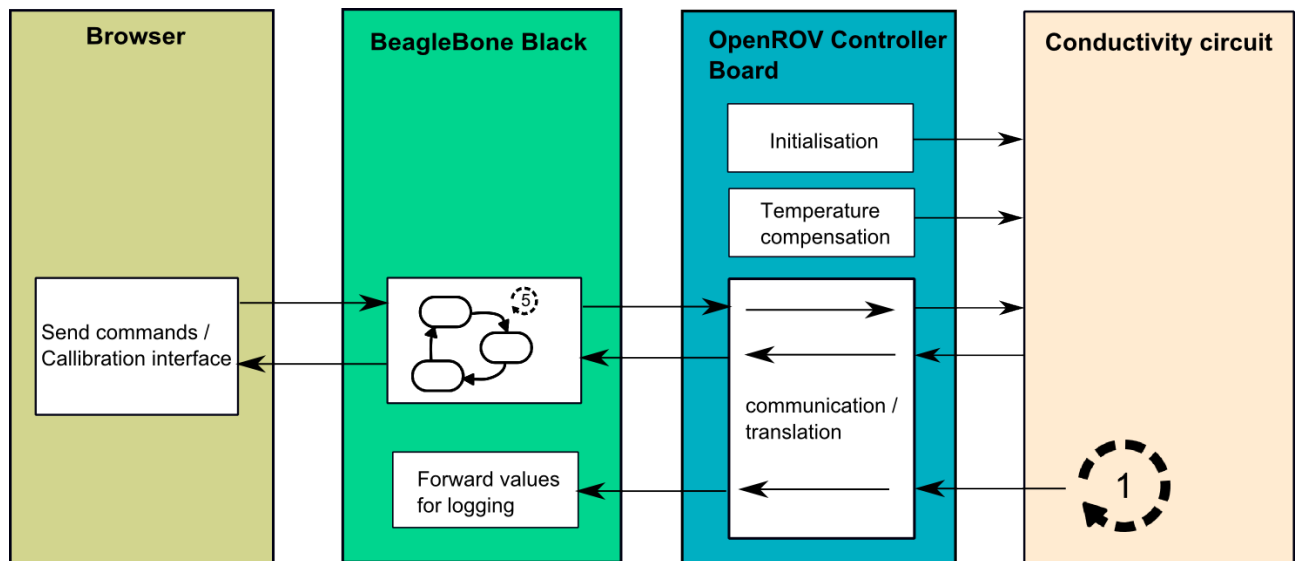


Figure 20: Software architecture of the conductivity measurement

The user can send commands and initiate a calibration procedure from the browser, which communicates with a state machine on the BeagleBone Black. This state machine handles all the logic and control the communication with the conductivity circuit. However, even if everything is controlled by the state machine, the messages to the conductivity circuit must pass through the OpenROV Controller board, that simply forwards those messages.

The OpenROV Controller board code, more than simply forwards messages, also initializes the conductivity circuit to ensure that it is in a proper functioning state for the system, and handles the temperature compensation for the conductivity measurement.

Values are also regularly received from the conductivity circuit and need to be forwarded to the BeagleBone Black code. This code, separately from the state machine, send the received values through a global event to be logged by the logging plugin.

### 6.3 Sensor

The chosen sensor to measure the conductivity is the ENV-40-EC-K0.1 from Atlas scientific. This particular sensor was chosen for two reasons. It is first fairly inexpensive which is in line with the project low cost aspect. The other reason is that a conductivity measurement circuit that derives the conductivity value from the signal sent by the probe is available and allows to gain a considerable time on the development, since such circuit are fairly complex to design.

Different probes are available, with different K coefficients. This coefficient correspond to the range of conductivity the probe is able to measure. the range of conductivity corresponding to each coefficient and the type of water those are used for are indicated in the table below :

K Coefficient	Conductivity range
K = 0.1	0.5 - 50 000 [ $\mu\text{S}/\text{cm}$ ]
K = 1	5 - 200 000 [ $\mu\text{S}/\text{cm}$ ]
K = 10	10 - 1 000 000 [ $\mu\text{S}/\text{cm}$ ]

A sensor of K = 0.1 has been chosen as the conductivity in the lake Erie, even if fairly polluted, is generally situated between 200 and 500  $\mu\text{S}/\text{cm}$  in lake erie [4]

A list of the material used for the conductivity measurement can be found in annexe I.2

## 6.4 Connections

The conductivity probe must be connected to the conductivity circuit, which is connected to the OpenROV Controller Board through 4 wires. 2 wires are used to power the circuit and the probe and the 2 other wires are used for the serial communication with the OpenROV. The probe is waterproof up to around 130m (1379 kPa of hydrostatic pressure) and can be submerged up to the BNC connector. This connector, as well as the conductivity circuit it is connected to, must then be waterproofed. The place available into the main electronic container being limited it is necessary to place it in an external waterproof case. The designed tube that contains this circuit is described in the section 11.4.

The figure 21 shows the different connections from the probe to the controller board, including to which OpenROV Controller board I/O the wires from the conductivity circuit are connected:

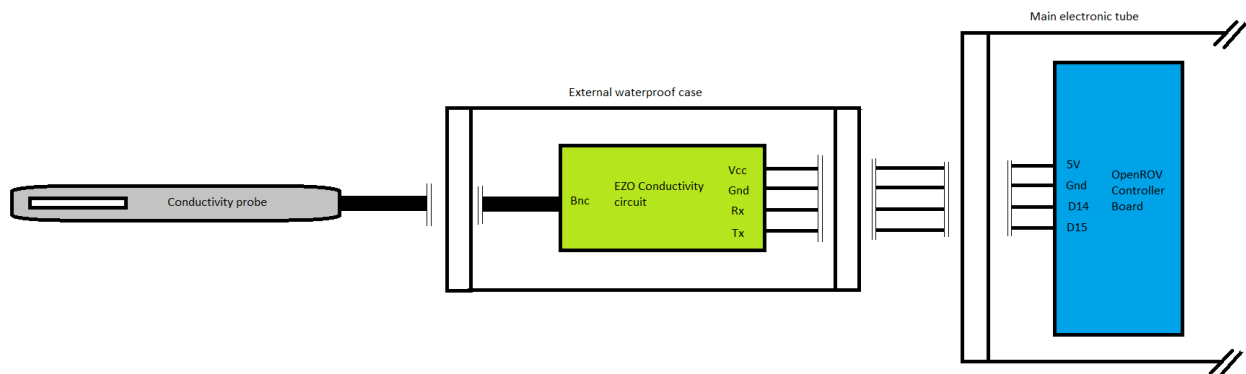


Figure 21: Connections for the conductivity measurement

The connections to the I/Os of the OpenROV Controller board are done through molex connector linked to wires sealed into one of the endcaps of the main electronic tube. This is described in the section 11.2 of this report.

See Annexe O for more information about the available I/Os of the OpenROV Controller board.

## 6.5 Signal and communication

The conductivity probe output a square signal that varies in function of the Conductivity measured. Those variations can be seen in the figure 22, that shows 2 measurements taken in two solutions with a different conductivity.

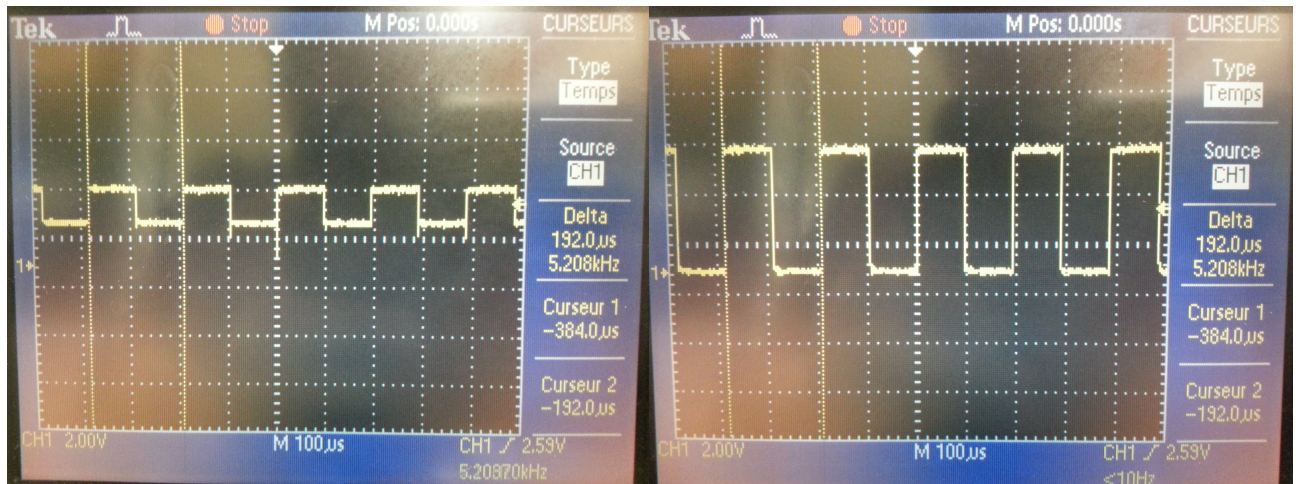


Figure 22: Conductivity probe output for 2 different conductivity

It was also measured that the signal received from the probe wasn't constant but was available only for half the period of a 1 Hz signal, as shown in the figure 23

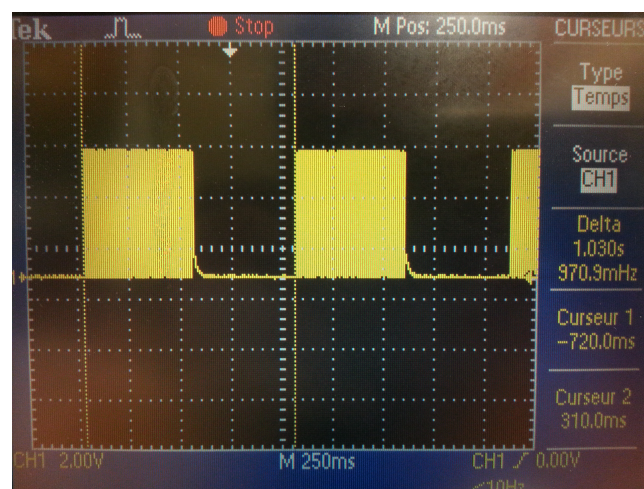


Figure 23: signal sent periodically

This signal is processed by the EZO conductivity circuit that is able to output the corresponding conductivity value and derive from it TDS, salinity and specific gravity of the water.

## 6.6 Conductivity circuit

To understand how the software implemented works, especially the state machine, the way the conductivity circuit communicates must be understood.

This circuit can understand various messages that can be received through RS232. When in the proper setting, the conductivity circuit will answer every messages with a confirmation.

Those confirmations always start with the character \* and can be :

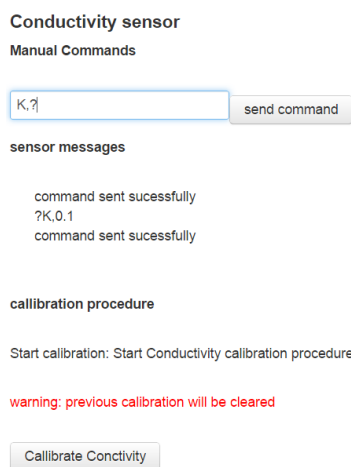


- ◆ \*OK if everything went well.
- ◆ \*ERR if the message was not recognized or couldn't be processed.
- ◆ \*RE, \*SL, \*WA, \*OV, \*RS, which are not used in this project.

The conductivity circuit can also answer some of the messages received. Those answers will always start with the character ?.

## 6.7 Browser software

When displaying the cockpit interface from the browser, the functionalities for the conductivity plugin are implemented into the menu `Diagnostic` on the upper-right part of the page. In this menu, the elements shown in figure 24 are visible and allow to handle the functionalities available for the conductivity sensor.



Commands can be sent using the `Manual commands` text-box. The format for the commands is the same that the one of the EZO Conductivity circuit. Under the text-box, events and messages from the sensor are displayed indicating if the commands are sent successfully.

A calibration procedure can also be launched from this interface using the button `Calibrate conductivity`, which will guide the user through various step for the sensor's calibration. When performing an action from the interface, messages are sent through websockets to the state machine that will handle them.

Figure 24: Web interface for the conductivity sensor

The calibration procedure's interface needs to be updated at each step of the calibration, displaying for each step new information about what to do. Those changes of display are always done when a confirmation of the previous calibration step realisation is received through the websockets.

## 6.8 BeagleBone Black software

### 6.8.1 State machine

A state machine is implemented in the `index.js` file of the conductivity plugin. This state machine has 2 purposes :

The first purpose is to handle the communication between the user and the sensor. The user might want to send commands directly to the sensor in order to modify a parameter or verify that the conductivity circuit is properly configured.

The second purpose, strongly linked to the first one is to handle the calibration procedure for the sensor. Even if this calibration can be done using standard commands, most users don't want to go through the sensor's data-sheet to find the calibration procedure and the commands that need to be sent. Thus, it has been necessary to guide the user through the procedure directly from the rover's interface. This procedure is implemented in the same state machine because it is strongly linked to the communication with the sensor and doing so allows to avoid unwanted behaviours if commands are sent while the sensor is being calibrated.

The implemented state machine is shown in the figure 25 below :

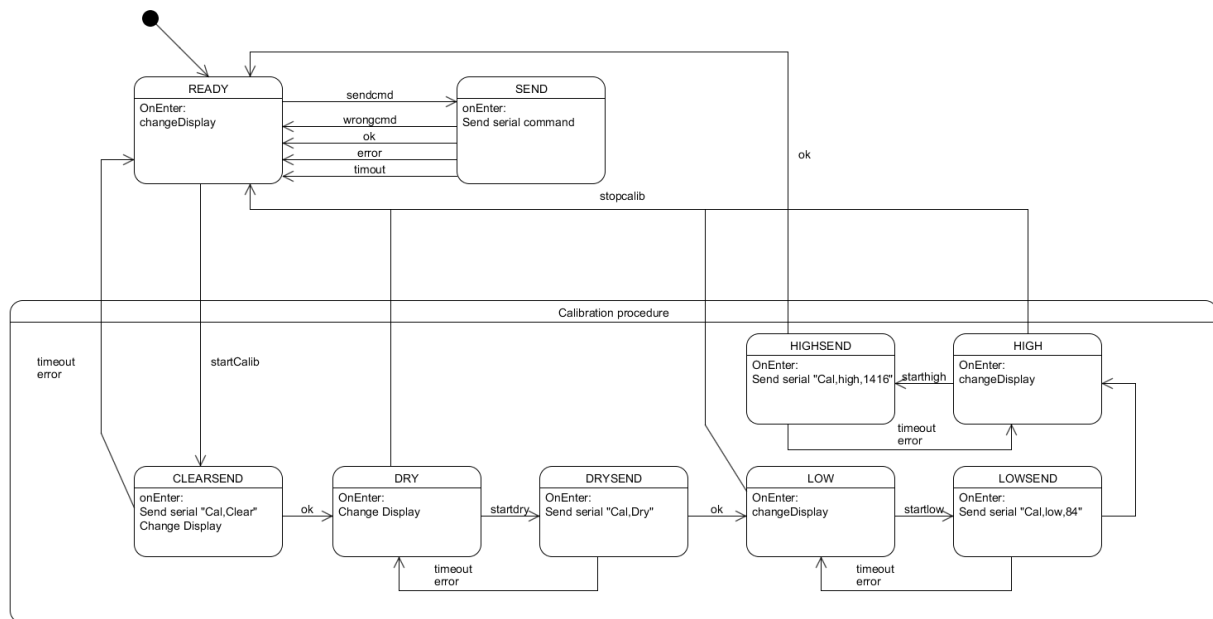


Figure 25: State machine for the index.js file

In this state machine, the 2 parts distinguished above can be clearly distinguished from each other. To make this distinction clearer, a global state `Calibration procedure` was added to the schema. This state is only here to make things clearer and is not directly implemented in the software.

When in the state `Ready`, the user can either send a message to the sensor or start a calibration procedure by interacting with the user interface.

If a message is sent, the machine will switch in the state `Send` and send the desired message to the sensor after checking if the message is known by the sensor. This verification is done through the function `cmdparsing()` in the `index.js` file. If this verification is successful, serial messages are sent to the OpenROV Controller board using the message format explained in section 4.5.

If the user starts the calibration procedure, the machine will simply progress into the global state `Calibration procedure` as the different calibration's steps are performed and finally return to the `Ready` state. In this case, the user will not be able to send a message (the 'manual commands' text-box is blocked in the browser when the calibration procedure starts).

When `change_display` is written in a state in figure 25, a message is in fact sent through the websocket to the browser, which handles the display.

When messages are sent, either with a manual command or during a calibration procedure, a timeout of 5 seconds is launched. If this timeout expires, an event `timeout` is sent and the serial message `condtm()` is sent to the OpenROV Controller board.



All events driving the state machine can come from 3 different sources :

The first source is a global event named `conc-event` sent by `OpenROVController.js`, which correspond to a confirmation from the conductivity sensor for a message sent (see section 6.6). This feedback will always be transferred to the state machine by the events `ok` or `err`. This can be seen in the figure 26 below :

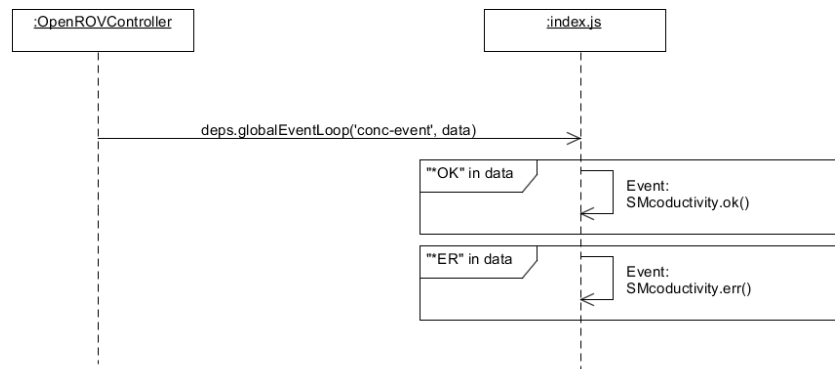


Figure 26: Events from sensor's answers

Another source of events is sourced directly from the `index.js` file, where the state machine is. These events handle problems in the communication when either the user sent a command that is not recognized or if a command has been sent but the sensor hasn't given answer after 5 seconds. Those events can be seen in the figure 27 below :

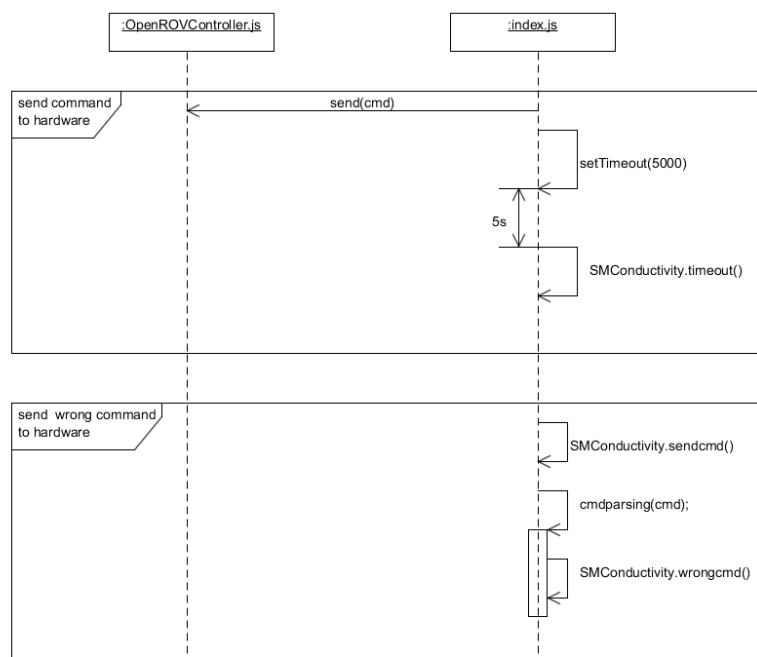


Figure 27: Events from index.js

The last source of events is the browser. Those events correspond to action wanted by the user, those can be message to be sent or calibration instructions. Those events can be send in the picture 28 below:

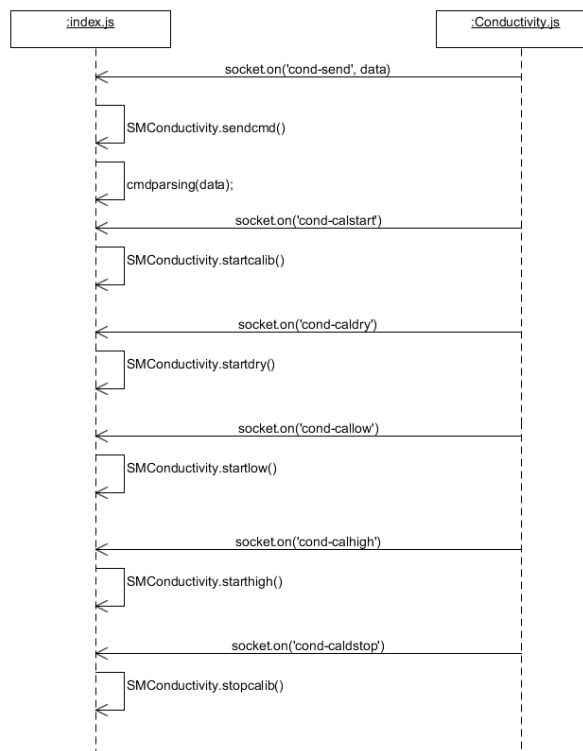


Figure 28: Events from the browser

### 6.8.2 Values forwarding

Values received from the conductivity circuit must be sent explicitly to the data-logging plugin. This is simply done by the following code in the file `index.js` of the conductivity plugin :

```

1  var tdsmodfactor = 1.13;
2
3  deps.rov.on('status', function(data) {
4
5      //if mesure obtained, share with all plugins (for sensor values display or logging)
6      if ('conr' in data) {
7
8          //recuperation of the conductivity sensor values
9          var parts = data.conr.split(',');
10
11          //the sensor should send 4 values, if not, this is not the right message
12          if(parts.length == 4) {
13
14
15              var sensordata = {
16                  conductivity: parts[0],
17                  tds: parts[1] * tdsmodfactor,      //tds value modified
18                  salinity: parts[2],
19                  specgrav : parts[3]
20              }
21
22              deps.globalEventLoop.emit('conductivity', sensordata);
23          }
24      }
25  });
  
```

This code is independent of the state machine implemented in the same file.

A subtle modification of the sensor values is made here: The conductivity sensor gives a TDS value that is derived from the conductivity by the following equation :

$$TDS = Conductivity * 0.53$$

However, the TDS value derived from the conductivity in lake Erie is known to be closer to  $TDS \approx Conductivity * 0.6$

The Conductivity circuit doesn't have any options allowing to change this factor, this needs thus to be done in a further processing. Several options are available : A factor can be added to correct the coefficient, which can be done either in the c++ or in the Javascript code. The TDS value can also be ignored and the value directly computed from the Conductivity value.

It has been chosen to modify the TDS value in the Node.js code because the Javascript code is easier to modify if necessary (no necessary compilation and upload). A better coefficient could be calculated for a specific location knowing the usual composition of the water.

To obtain a coefficient of 0.6, an additional factor of 1.13 has simply been added to the TDS value :

$$TDS_{LAKE} = TDS_{SENSOR} * 1.13 = Conductivity * 0.53 * 1.13 \approx Conductivity * 0.6$$

## 6.9 OpenROV Controller board software

### 6.9.1 Architecture

The functionalities implemented in the file `conductivity.cpp` handle mainly the low level communication with the sensor and some functionalities related to it. Those functionalities are:

- ◆ The serial communication with the sensor.
- ◆ The initialisation of the sensor.
- ◆ Handling the communication between the BeagleBone Black and the Conductivity circuit.
- ◆ Compensate the conductivity measurement depending of the temperature.

The figure 29 shows how the conductivity loop in the OpenROV Controller board is executed.

It can be seen that various variables are used for the control of this loop :

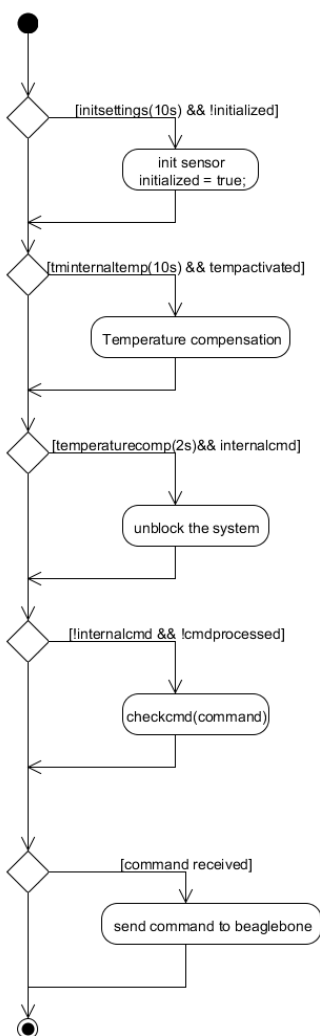


Figure 29: UML activity diagram for `conductivity.cpp`

- ◆ The variable `cmdprocessed` allows to control that only one command is sent at a time to the conductivity circuit. Even if this shouldn't append since the state machine implemented in the BeagleBone Black handles this functionality, it was done as a security since multiple commands sent at the same time (causing multiple confirmations from the sensor) could disrupt the good functioning of the state machine.
- ◆ The variable `internalcmd` controls if a command that wasn't sent from the BeagleBone Black was sent to the Conductivity circuit. This allows to avoid that the confirmation for the current command is sent to the BeagleBone Black, which would create unexpected events that might disrupt the state machine.
- ◆ The variable `tempactivated` allows to enable or disable the temperature compensation of the sensor.
- ◆ The variable `initialized` allows to execute initialisation commands for the conductivity circuit only once.

3 timeout are also implemented :

- ◆ The timer `initsettings` counts to 10 seconds before sending the initialisation commands to the Conductivity circuit. This timer is started in in the function `device_setup()`
- ◆ The timer `tminternalcmd` handles a timeout of 2 seconds for the internal commands (since these are not controlled by the state machine timeout).
- ◆ The timer `temperaturecomp` that allows a temperature compensation every 10 seconds. This timer is started in in the function `device_setup()`

### 6.9.2 Serial communication

The communication with the conductivity circuit uses a RS232 protocol, which is handled by the object `Serial3` in the arduino library. The baudrate must simply by set at 38400 to be able to communicate with the conductivity circuit by using its begin method (`Serial3.begin(38400);`) in the function `device_setup()`.

Sending commands to the conductivity circuit can be done by simply using the method `print(String)` of the serial object.

Receiving commands is however slightly more complicated, since it has to be received character by character, and is illustrated in figure 30. This reception is done through the method `serialEvent3()` which receive characters and add them one by one to a string until it receives a carriage return. In this case an boolean is updated indicating that the string is completed and can be used by the `device_loop`

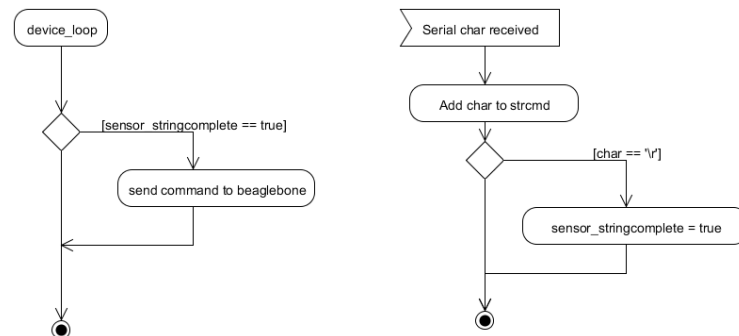


Figure 30: Reception of serial strings

The base of the code handling the serial communication was found in a document from atlas scitienfic [10]

### 6.9.3 Sensor initialisation

The conductivity sensor must be initilized in order to ensure that it is in a proper fonctionning state. As it can be seen in figure 29, this initilisation is not done in the function `device_setup()` but 10 seconds after. The reason for this choice is that it must be ensured that the conductivity circuit has properly booted and is ready to accept commands.

The command send at initialisation are :

- ◆ `Response, 1` which makes sure that the conductivity circuit is giving a feedback for all commands it receives. Which is indispensable to generate the proper event for the BeagleBone Black's state machine.
- ◆ `K, 0.1` which makes sure that the conductivity circuit is configured to use the right probe.

### 6.9.4 Communication with the sensor

Serial strings received from the BeagleBone black must be translated into the commands understood by the conductivity circuit. This translation functionality is implemented in the method `checkcmd(Command command)`. Response from the Conductivity circuit must also be sent to the BeagleBone Black by parsing them into the right format for the communication between board.

The messages that can be send to the Conductivity circuit and how they are translated for the serial communication between the BeagleBone Black and the OpenROV Controller board can be seen in annexe C

### 6.9.5 Temperature compensation

Aside from the translation of messages, it is necessary to adjust the temperature compensation of the sensor regularly. This is done in the method `changetemperaturecomp()`. This method simply get the current temperature from the temperature sensor and sent a command to the Conductivity circuit in order to correct the temperature ajustement for the conductivity measurement.

The value of the temperature is obtained from the variable `envdata::TEMP` declared in the file `device.h`. This variable correspond to the temperature measured by the IMU/Depth sensor.

## 6.10 Communication between circuits

The communication from the OpenROV Controller board to the BeagleBone Black needs some precision for this sensor. As explained in section 4.5.2, the messages from the OpenROV Controller board are forwarded to the Node.js global event loop through the script `OpenROVController.js`, but what is implemented causes problem for one-time events.

The `OpenROVController.js` must be modified by sending a few event sent aside of the status event. The added code is the following :

```

1     hardware.on('status', function (status) {
2         if ('comm' in status) {
3             globalEventLoop.emit('comm-event', status.comm);
4         }
5
6         if ('conc' in status) {
7             globalEventLoop.emit('conc-event', status.conc);
8         }
9     }

```

`comm` corresponds to a response from the sensors to a request from the browser and provide information that need to be displayed for the user.

`conc` corresponds to a feedback from the sensor about how the last command could be executed. it can be `*OK` or `*ERR`, which are important events for the proper functioning of the state machine.

## 6.11 Calibration

The calibration of the conductivity sensor is made through the the Conductivity circuit. This circuit allows multiple commands in order to calibrate the linked sensor in 2 different ways :

- ◆ Calibration on a single point
- ◆ Calibration on two points

The calibration on a single point is less precise over a large range and don't cover the entire range of the sensor. This calibration could be enough for the use in lake Erie since the conductivity is not likely to vary considerably[4].

However, If the rover is used in other environments a single point calibration could be inappropriate. For this reason it has been chosen to facilitate a 2 point calibration in the software for a standard user. An advanced user is still able to calibrate the sensor manually by sending manual commands and thus be able to calibrate the sensor on a single point only.

The commands for the calibration are the following :

command	functionality	answer
Cal,?	Ask for the actual calibration	0   1   2
Cal,clear	Clear actual calibration	-
Cal,dry	Calibrate the dry sensor	-
Cal,one,n	Calibrate the sensor in a calibration solution of n $\mu\text{S}/\text{cm}$	-
Cal,low,n	Calibrate the sensor in a calibration solution of n $\mu\text{S}/\text{cm}$ for the low point of the calibration	-
Cal,high,n	Calibrate the sensor in a calibration solution of n $\mu\text{S}/\text{cm}$ for the high point of the calibration	-

Those command have to be used in the following order for a 2 point calibration : clear, dry, low, high.

And in the following order for a single point calibration : clear, dry, one.

The commands can be sent from the cockpit interface as explained in section 6.7. The guided 2 points calibration procedure can also be found at the same place is only valid with calibration solutions of  $84 \mu\text{S/cm}$  and  $1414 \mu\text{S/cm}$ .

## 6.12 Tests

### 6.12.1 Method

Different tests were made. At first, the sensors were compared over a large range of conductivity. The sensors were then compared by taking several measurement using a single solution. Finally, test were done directly in the lake where the values of all sensors were recorded

The Values returned by the conductivity sensor were tested in the range it would be used. To do this test, an other conductivity sensor, an Hanna instruments HI 8733 was used for comparison. This sensor was calibrated with a  $1'413 \mu\text{S/cm}$  solution.

Since the sensor is able to give also the value of TDS, the values returned have been also compared to an other TDS sensor, an Hanna instruments HI 8734. This sensor was calibrated with a  $800 \text{ mg/l}$  solution.

### 6.12.2 Conductivity over range

The conductivity was measured over a large range using distilled water into which NaCl was progressively added in order to increase its conductivity. Measures were then taken using all different sensors in order to make a comparison. The results, represented in figure 31 and 32, show that in the lower part of the measured range, results are similar. However, they start to slightly deviate from a certain value for both TDS and Conductivity.

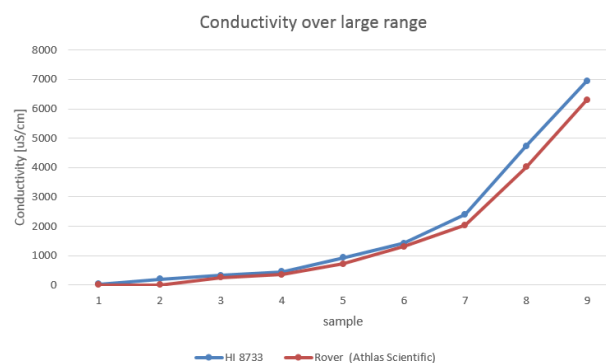


Figure 31: Conductivity measured with several samples using both the Atlas scientific sensor and the HI 8733

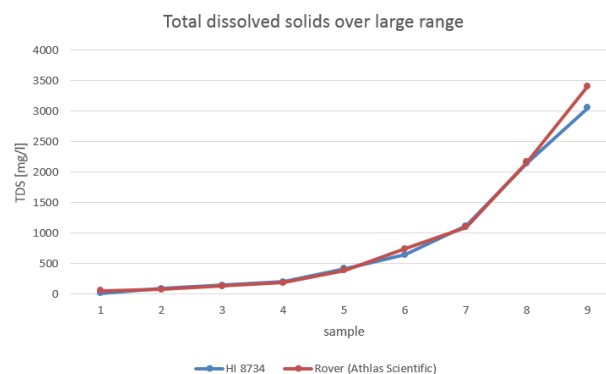


Figure 32: Total dissolved solids measured with several samples using both the Atlas scientific sensor and the HI 8734

This deviation is however not a problem for the use of this sensor since Conductivity values measured in lake Erie would not reach such high values, the conductivity in lake Erie being generally between 200 and 500  $\mu\text{S}/\text{cm}$  [4].

The deviation is also likely to be due to the reference sensors, that are calibrated around only one value against two for the Atlas scientific sensor, and will thus be less accurate far from their calibration value. The best way to check this would be to use a high certified conductivity solution, and compare the sensors with it. Such high conductivity solution were not available during this project, reason for which this hasn't been done.

### 6.12.3 Precision test

The precision of the sensor has been tested and can be compared to the references sensors. For each sensors, the mean value and the standard deviation have been plotted based on a dataset of 8 measures by using Octave. the script that has been used can be found in F.

The measures were taken using alternatively the 3 sensors in the same solution. Measurements were not taken simultaneously to make sure that the magnetic field emitted by the conductivity sensors would not influence each other. The sensors were left in the solution for about 30 seconds before recording the value.

The results are the following for the conductivity :

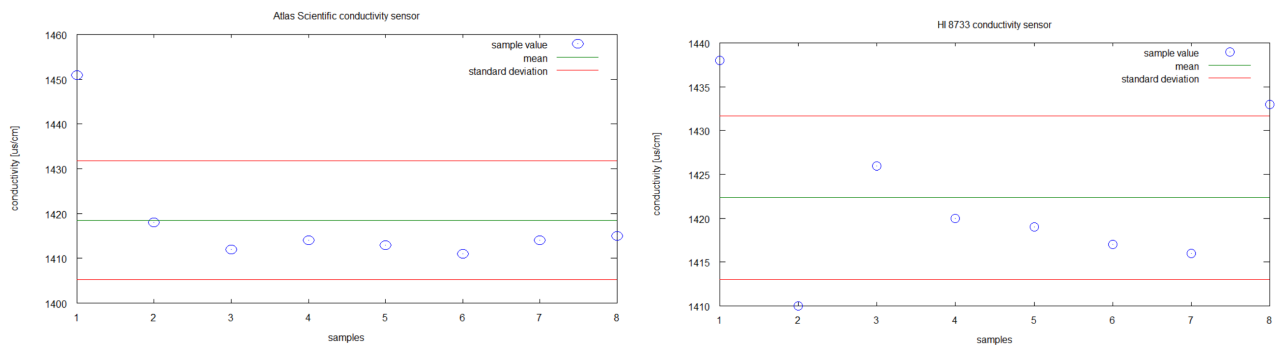


Figure 33: Comparison Conductivity sensors

the Standard deviation were of  $\pm 12 \mu\text{S}/\text{cm}$  for the ENV-40-EC-K0.1 and  $\pm 9 \mu\text{S}/\text{cm}$  for the HI 8733

The results are the following for the TDS :

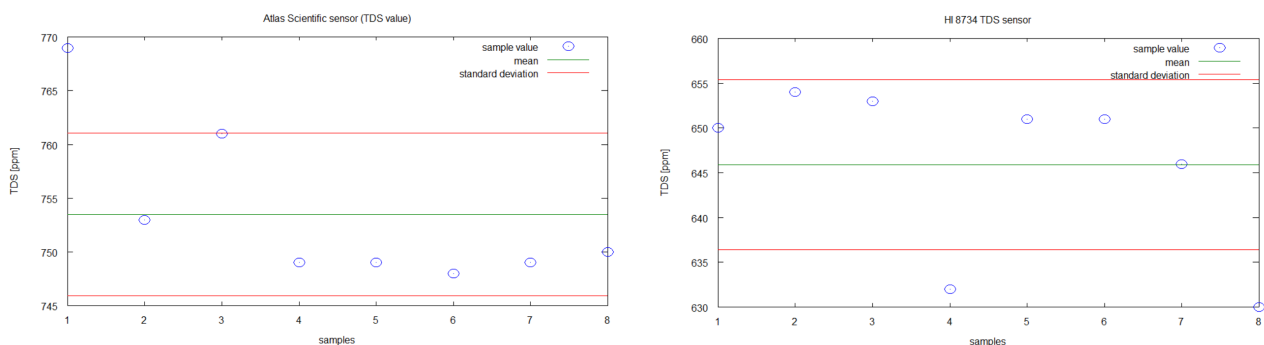


Figure 34: Comparison TDS sensors

the Standard deviation were of  $\pm 8 \text{ mg}/\text{l}$  for the ENV-40-EC-K0.1 and  $\pm 9 \text{ mg}/\text{l}$  for the HI 8734



#### 6.12.4 Field measurements

Measurements have been taken at different locations between Cleveland and Sandusky. Values of the Conductivity and TDS were recorded by the rover's conductivity sensor and the Hanna instruments TDS(HI 8734) and Conductivity(HI 8733).

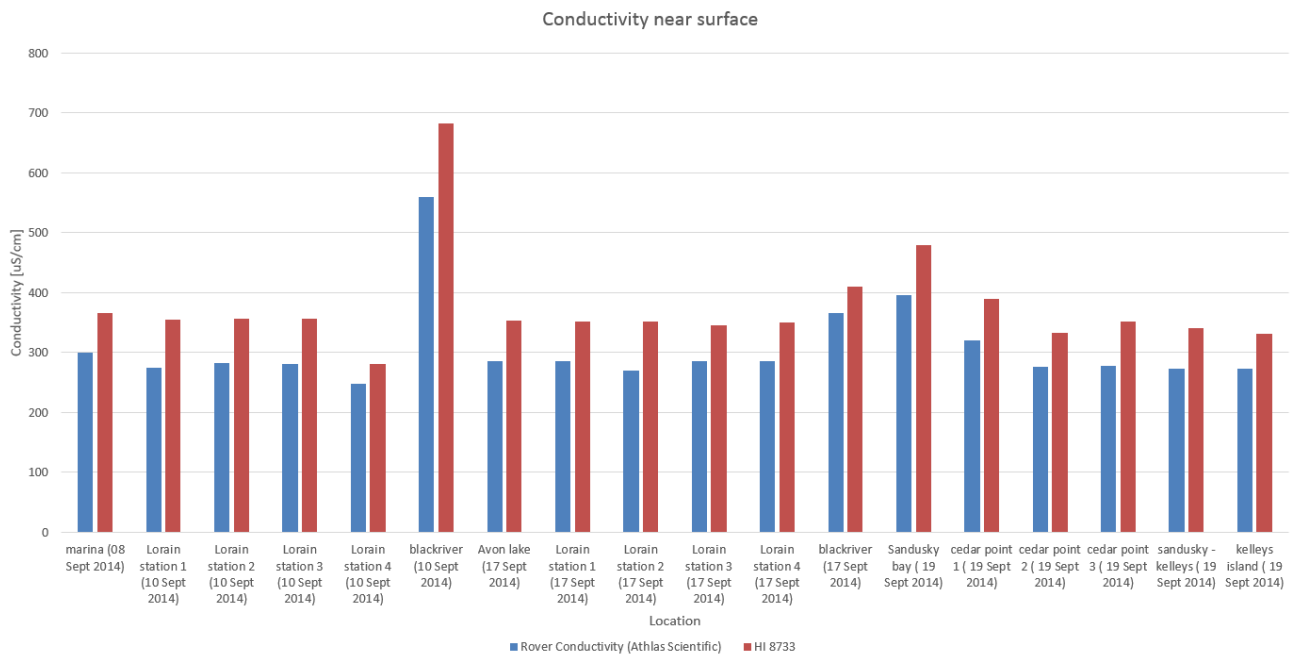


Figure 35: Conductivity measured at several locations using both the Atlas scientific sensor and the HI 8733

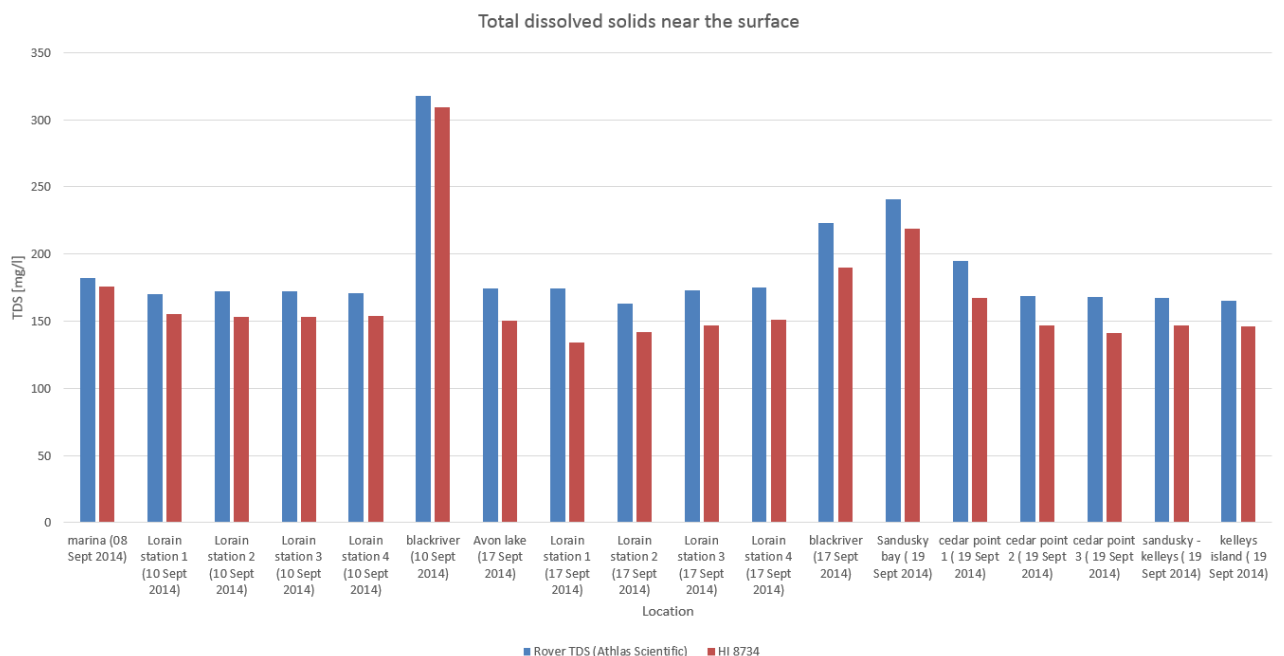


Figure 36: TDS measured at several locations using both the Atlas scientific sensor and the HI 8734

The values measured show a consistence between all values, but are always slightly different. However, the values can here only be compared qualitatively, because of the calibration of the Hanna instruments sensor that was done with only one solution with a much higher value than the measures.

### 6.13 Conclusion

The conductivity measurement was implemented entirely and is properly working. The test concerning its accuracy and precision were satisfying over the entire used range.

The communication with the sensor can however be ameliorated. In rare cases, messages sent from the browser will not be received by the sensor and a timeout will be triggered. This problem doesn't block the system but can decrease the user's experience.

The software has also not been entirely tested, which should be done in order to make sure that no bugs are left. One of those tests should be to write test code for the state machine to ensure that no bugs appear for whatever cases.

## 7 TEMPERATURE SENSOR

### 7.1 Introduction

#### 7.1.1 Temperature

The temperature of water is a very important water quality parameter, it might not seem so at first glance but it strongly affects the wildlife and chemical reaction in the water [3][8]. Temperature governs which species are present in the lake, and if temperature changes too much, their population might lower and even disappear.[8]

Many factors might change the temperature in lakes. Daily changes occur at the surface of the lake during the cycle day/night and seasonal changes can be seen in the entire water column. A phenomenon of thermal stratification might develop throughout the year, this means that different layers of the lake might have different temperatures at different depths and not mix. [8] Those different layers might also have different other properties, such as electrical conductivity or Dissolved oxygen (DO).

Those properties are in fact directly affected by the temperature. As the temperature increases, the water is able to dissolve more minerals which will increase the electrical conductivity of the water.[7]. Another parameter that will directly be affected is the quantity of Dissolved oxygen, that warm water is able to hold less than cold water. The water may then be saturated with oxygen but still not contain enough for survival of aquatic life[8][7].

Aside from the water quality, there is another reason to measure temperature, which is the temperature compensation for other sensors. Many sensors need a temperature compensation to output accurate values, which is the case in the rover set-up for the pressure sensor and the conductivity sensor.

#### 7.1.2 Sensing solution

A measure of temperature can already be done by the OpenROV using the temperature sensor of the IMU/Depth sensor. However, this sensor is very slow to get to a location's temperature. For this reason it was chosen to use an alternate solution able to take quick measurements, which is an interesting feature for a mobile sensing platform.

### 7.2 Architecture

The general architecture for the temperature measurement is as shown in figure 37 below:

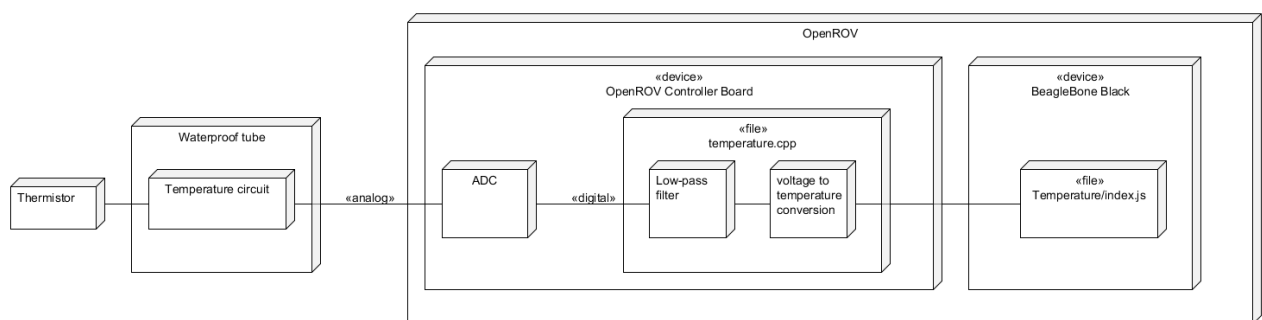


Figure 37: Architecture for the temperature measurement

The probe is a thermistor which must be driven by a circuit placed into a small waterproof container. This circuit outputs a voltage between 0 and 4 volts and is connected to the OpenROV Controller board through an ADC converter.

The signal obtained from the ADC suffers from IEM noises, that must be filtered. Once filtered, the value obtained needs to be converted into an equivalent temperature. The filter and the conversion are both made into the file `temperature.cpp` of the OpenROV Controller board.

The temperature value is then transferred to the Beaglebone black and its plugin `Temperature`, that simply forward the temperature value to all other plugins.

A block diagram of the software architecture for the temperature measurement can be seen in the figure 38 below :

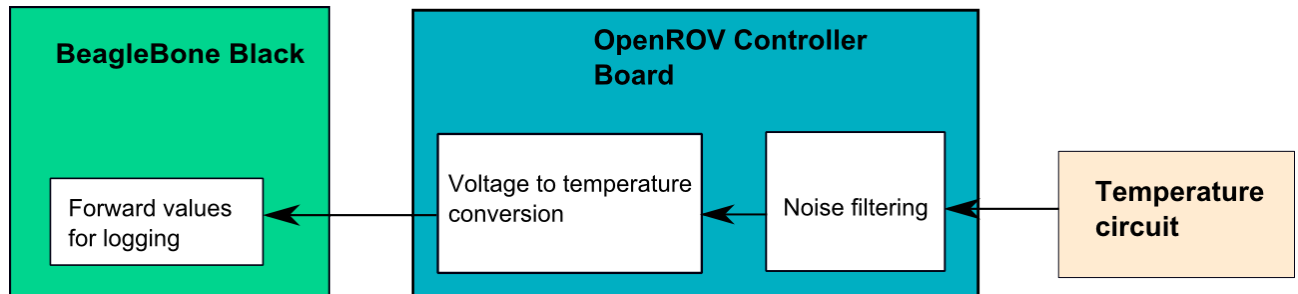


Figure 38: Software architecture of the temperature measurement

It can be seen in the picture above that all messages go from the sensor to the BeagleBone Black. No communications goes to the other way.

### 7.3 Sensor

A important thing to take into consideration when choosing a sensor is the mobile aspect of the system. The rover will move into water, which means that the temperature might change rapidly during a measurement, either because of a cold current in a specific location or because of the rover going deeper where water is often colder. It is then interesting to have a sensor with a thermal constant as low as possible.

Unfortunately, most affordable sensors on the market are either not able to work underwater or have a very high thermal time constant. The temperature sensor included with in the IMU/Depth sensor, was tested in order to define its thermal time constant. This test was done with a drop of 4 °C, with a measured constant of approximately 30s (figure 39), which is way too slow for a fast changing environment.

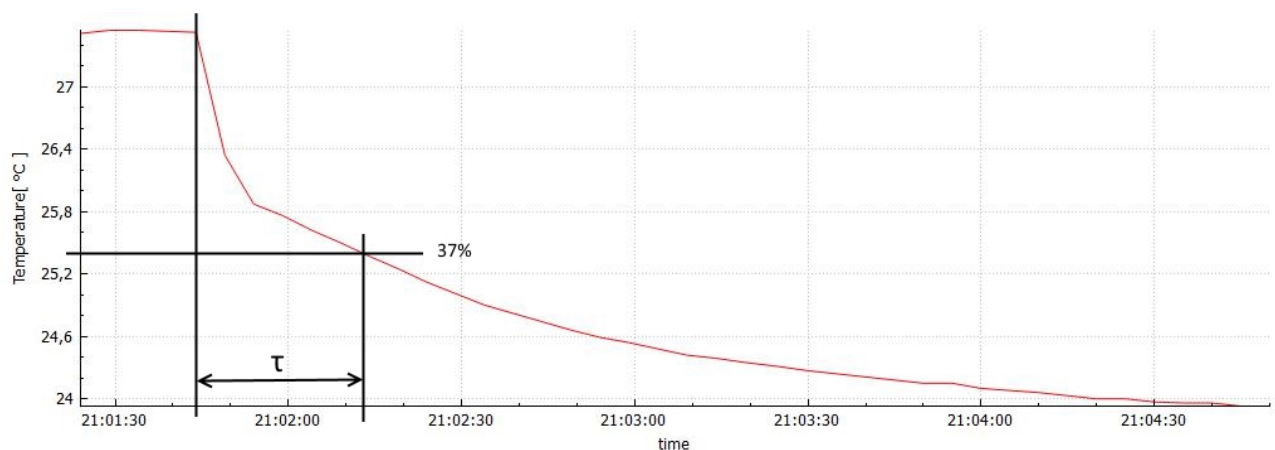


Figure 39: Measure of the IMU/Depth's temperature sensor's thermal time constant

The conclusion was made that another sensor should be found, as small as possible in order to optimize this time constant. For an optimal result the choice was reduced to the most basic components available for temperature measurements : Thermistors, Thermocouple and RTD's.

the advantages and disadvantages of each of those component were taken into consideration and are summed up in the following table :

sensor	Advantages	Disadvantages
Thermistor	Very small, Quick response time, Cheap, Accurate	Fragile, narrow range
Thermocouple	Robust, Large range	Accuracy, Low signal strength
RTD	Linearity, Stability	Expensive, Slow response time

It can be deduced from the table above that the `thermistor` is the best choice in this case. Being very small its thermal constant is thus reduced and allow a very quick response which is as explained earlier is the important characteristic. High accuracy thermistors are available on the market for very low prices, which allows to quickly change the thermistor without specific calibration. This compensate its natural drawback of being quite fragile due to its small size. The disadvantage of having a non linear response over a large temperature range is as well a minor problem, the resistance to temperature conversion is implemented via software which allow to do those calculation without problem. Another potential problem is its narrow measuring range. However, the temperature measurement in the lake will never be under 0 °C or above 50 °C which is a narrow enough for a thermistor measurement.

## 7.4 Connections

The driving circuit needs to be connected to the OpenROV Controller board to an ADC input, which was chosen to be the AN14. The circuit needs also a ground reference and a 5V power supply.

Those connections can be seen in the figure 40 below:

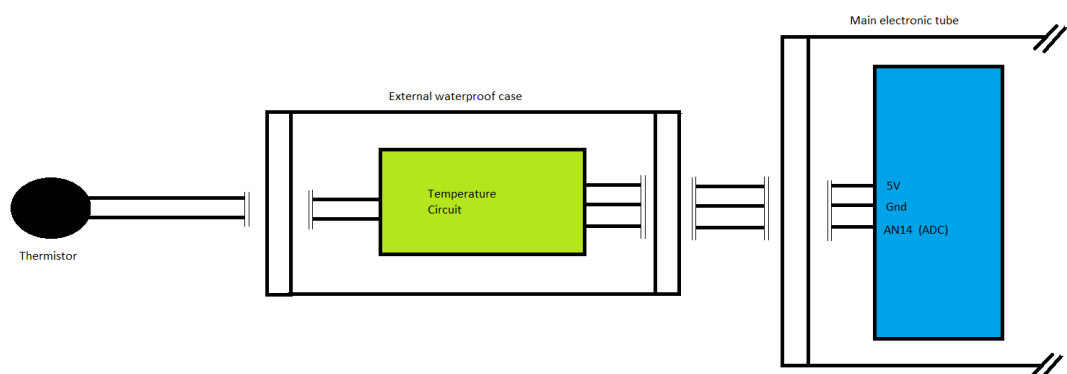


Figure 40: Connections for the temperature measurement

## 7.5 Thermistor Circuit

### 7.5.1 Analyse

The major problem of using a thermistor is the necessity to build a circuit to drive it. The circuit must be able to output a voltage from the the changing resistance of the thermistor.

A first way to do it is to use a current source that can provide a constant current through the thermistor. The voltage on the thermistor can then be output by using an operational amplifier. This method is a good way to get the thermistor value. An alternative, that was chosen is to use a voltage reference instead of a current reference and to get the thermistor's value through a voltage divider. The reason for this choice is essentially its lower price (the voltage reference is a lot cheaper than a current reference). A Wheatstone bridge could be used instead of a voltage divider, but because of the large range of the thermistor's values, the sensitivity would becomes way to small.

### 7.5.2 Circuit

The circuit designed for the thermistor is represented in the figure 41 below :

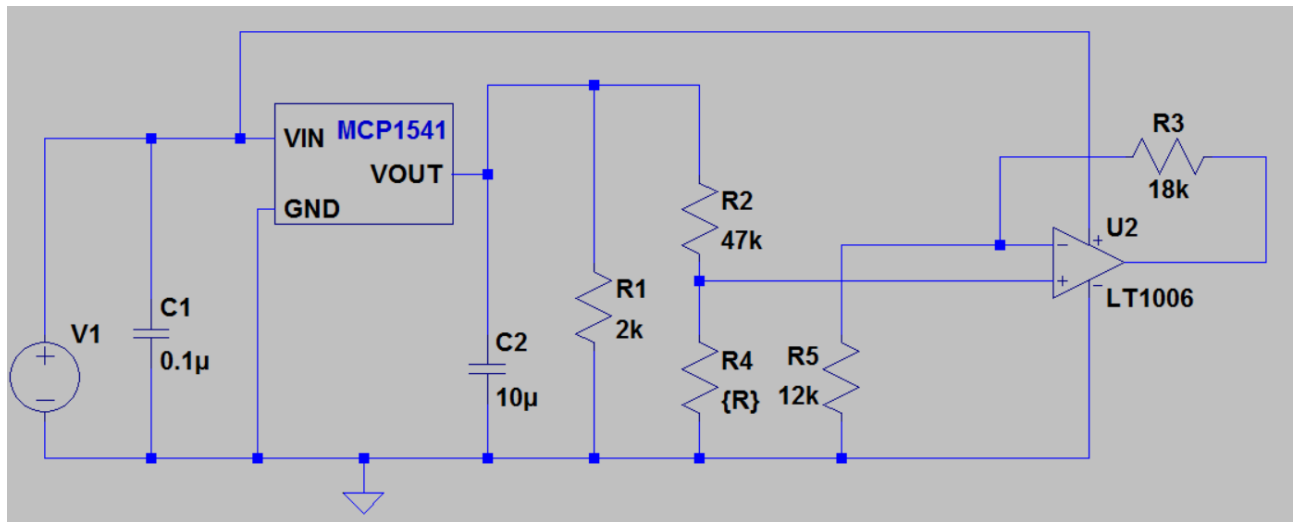


Figure 41: Circuit for the measurement with the thermistor

It is separated in 3 parts :

The first part is a voltage regulation. To avoid to use an external power source, the circuit is directly connected to a 5V pin of the OpenROVController board. This source is however not very stable and a very precise power source is necessary for a proper measurement.

The second part is the simple voltage divider with one of the resistance being the thermistor.

The last part is a non-inverting amplifier, in order to obtain an output range as large as possible for the Analog-digital converter to which the circuit will be connected.

Those 3 parts are explained separately and in detail in the following sections.

The material needed for the temperature measurement can be found in annexe I.3

### 7.5.3 Voltage regulator

The power source for the circuit is obtained from the OpenROV Controller board. However, this source is highly unstable as was measured with oscillation reaching  $\pm 0.3$  V, which is not acceptable for precise measurements.

The solution is to use a precision voltage reference which would give a stable signal for the thermistor to be measured. The chosen reference is a Microchip MCP1541.

A zener diode could be used for a similar use but the MCP1541 has the advantage of a better stability across time and temperature ( $\pm 50$  ppm/ °C). The temperature stability is an important criteria since the circuit is for an outside use and might be used during all seasons.

The part of the circuit concerning the voltage reference is the following :

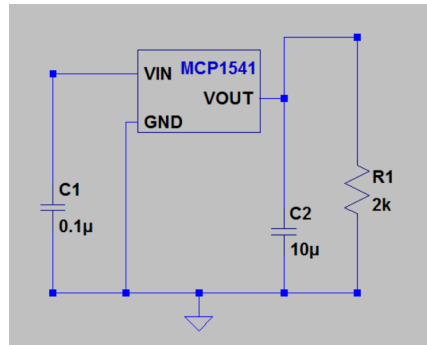


Figure 42: Voltage regulation in the circuit

The Capacitor C1 serves as a decoupling capacitor for the power supply.

The Capacitor C2 is required by the MCP1541 in order to stabilize the voltage reference. The required capacitor must be between 1 uF and 10 uF. Although a 10uF capacitor was used, alternatives were measured without noticeable differences.

The resistance R1 was added in order to allow the MCP1541 to work properly in the design. When tested the circuit was initially not working because of the very high resistance of the voltage divider. The output current drive is indicated on MCP1541 data-sheet of  $\pm 2\text{mA}$ . A resistance of  $2\text{ k}\Omega$  has been calculated.  $R1 = V_{OUT}/I_{OUT} = 4.096/0.002 = 2048 \approx 2\text{ k}\Omega$

#### 7.5.4 Divider

The measures are done through a simple voltage divider, one of the resistance being the thermistor. It is however important to take a few elements into consideration :

- ◆ The thermistor must not self heat, and thus it has to dissipate a maximum of 0.1mW.
- ◆ The value of the thermistor can vary from  $3\text{ k}\Omega$  to  $30\text{ k}\Omega$ , which represent a range from 0 to  $50\text{ }^{\circ}\text{C}$ . Temperatures that will never be exceeded by taking measurement in lake Erie.
- ◆ The output should not go above 5 volts in order to be connected to the analog-digital converter of The OpenROVController board.

Using a resistance of  $47\text{ k}\Omega$  with the thermistor, minimum and maximum total resistance in the divider are of respectively 50 and  $77\text{ k}\Omega$ . This makes a minimum and a maximum current of:  $I_{max} = V_{out}/R_{min} =$

$$4.096/50000 = 81.92\mu\text{A}$$

$$I_{min} = V_{out}/R_{max} = 4.096/77000 = 53.19\mu\text{A}$$

The Power dissipated in the thermistor is then :  $P_{max} = R_{max} *$

$$I_{min}^2 = 84.9\mu\text{W}$$

$$P_{in} = R_{min} * I_{max}^2 = 20.13\mu\text{W}$$

The limit of 0.1 mW is not exceeded which avoids any self heating.

Using those values, the output range of the divider is represented in the figure 43 below :



Figure 43: LTSpice simulation of the output of the voltage divider in function of the value of the thermistor resistance

### 7.5.5 Amplifier

The signal at the voltage divider reaches a maximum of only 1.6V. The range can be amplified to cover a maximum of the range of The OpenROV Controller analog-digital converter.

To do it, using an operational amplifier is the best option. However most of them require a dual power supply, which is not directly available. It is possible to create a circuit able to transform the single supply at the entry into a dual supply for the amplifier but a better and simpler option that was chosen is to use a single supply amplifier.

Single supply have however often problems with very low voltage input and output, which often doesn't extend to the ground. It was necessary to find one that is able to manage an input of at least 0.25V (minimum output from the voltage divider) without having any offset at the output.

The chosen amplifier is a a LT1006, which is a precision single supply op amp that fits all required characteristics and can be supplied with the available 5 volts. It has however a very low saturation point, measured at 4.1 volts, when supplied with 4.7 volts (minimum measured voltage of the 5v pin that supply the circuit)

4 volts were chosen as being the maximum output for the amplifier. The maximum output of the divider being 1.6 volts, a gain of 2.5 has been chosen.

$$G = \text{MaxOutput} / \text{MaxInput} = 4 / 1.6 = 2.5 \quad (1)$$

This gain can be obtained by a non-inverting circuit for the operational amplifier, as shown in the figure 44 below.



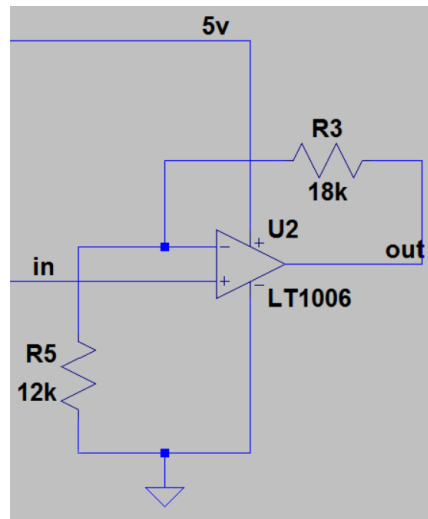


Figure 44: Voltage regulation in the circuit

The input signal (output of the voltage divider) is connected to the positive input of the amplifier, and the gain is fixed by the resistance R3 and R2 :

$$G = 1 + R3/R2 = 1 + 18/12 = 2.5 \quad (2)$$

The output of the operational amplifier corresponds to the output of the circuit and can be connected to the Analog-digital converter.

## 7.6 Signal filtering

### 7.6.1 Design

The Temperature circuit, once connected to the OpenROV Controller board is affected by a random noise as it can be seen in the picture 45 below :

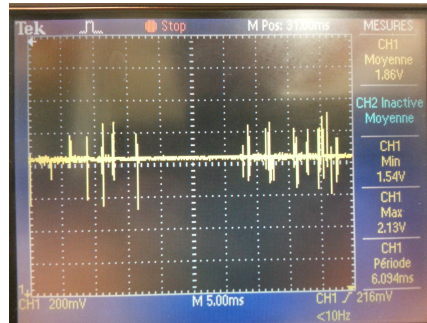


Figure 45: Noise between the temperature circuit and the OpenROV Controller board

This noise is due to electromagnetic interferences coming from the engines, the servomotor and the conductivity probe. Since those interferences affect the result of the measure it is necessary to filter it.

A solution could be to place a small circuit implementing a low pass filter and a common mode filter before being processed by the analog-digital converter of the OpenROV Controller board. Another solution is to implement a digital filter in the software after the Analog-digital converter. This last solution was chosen for 2 reason :

- ◆ A digital filter is cheaper and doesn't need any additional material, which is an important factor for this project.
- ◆ The place available into the electronic tube is very limited, thus, additional circuit are difficult to add.

The temperature signal measured is almost continuous and the noise to filter is random, as it can be seen in the figure 45. The filter must then be a low-pass filter with a cut-off frequency as low as possible.

The simplest filter that fits those condition is a moving average, which is the simplest FIR low pass filter possible. Despite its simplicity, it is optimal when it comes to filter random noises that have a Gaussian distribution. This filter also avoids sharp changes, which is pleasant in this case because of the extreme sensitivity of the thermistor. It does however have a strong delay directly due the the number of samples taken and the sampling frequency.

The equation characterizing this filter is the following :

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} (x[i + j]) \quad (3)$$

The delay of the filter is in the current design not a significant problem since data are registered every 5 seconds, it must simply not go past this limit. The filters delay depends on the sampling frequency and the number of samples. It seems reasonable to have a delay of about half of the logging time. An amount of 50 samples for a sampling frequency of 20Hz seems to be a reasonable choice.

$$delay = \frac{1}{Samplingfrequency} * Numberofsamples = \frac{1}{20} * 50 = 2.5seconds$$

The sampling frequency should be raised if the logging interval is reduced in the future.

### 7.6.2 Simulation

Simulations of the filter were made using Octave in order to check its functionality. A dataset of 140 samples was used for this simulation for an ambient temperature, which corresponds to a signal such as the one shown in the figure 45. The simulation file used for the following simulation can be found in annexe G

The plotted dataset, as represented in the figure 46 can be seen to have a large dispersion of result, having for consequence a very low precision. Once filtered, this signal becomes way more precise but is impacted by a delay of 50 samples as it can be seen in the figure 47.

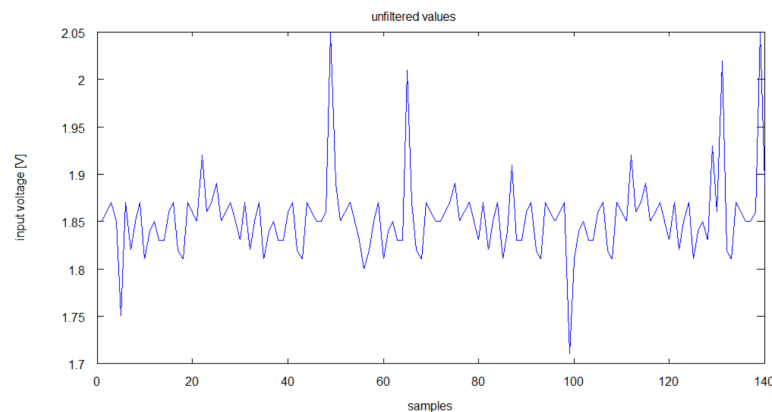


Figure 46: Input data before filtering

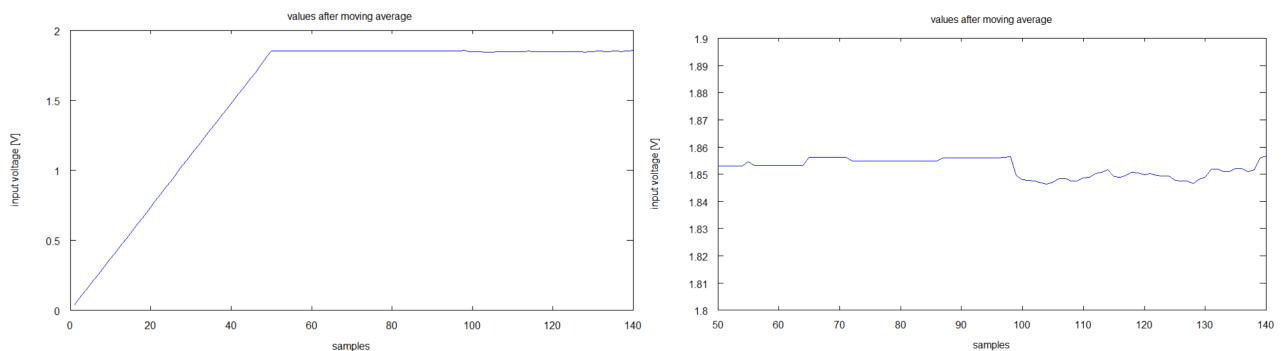


Figure 47: Data after filtering

The samples before and after filtering can be seen in the figure 48. The dispersions of the samples is noticeably increased and dispersed around the mean as planned. The standard deviation is lowered from  $\pm 0.043$  V to  $\pm 0.003$  V if the 50 first samples are ignored.

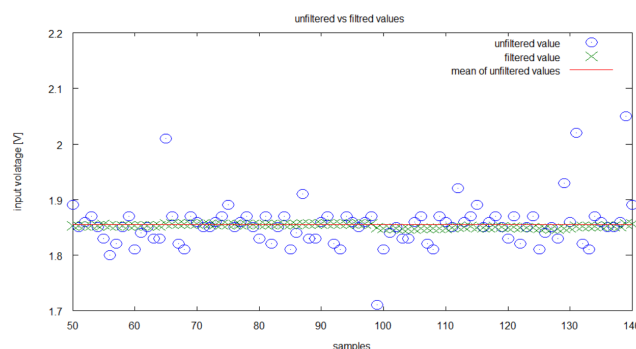


Figure 48: Samples before and after filtering

## 7.7 OpenROV Controller board software

### 7.7.1 Analog-Digital conversion

The first functionality that needed to be implemented is the analog-digital conversion. This conversion is done almost automatically with the Arduino library. The Pin on which the circuit is connected must be simply declared in the `temperature.cpp` header has follow :

```
1 Pin temp("temper", A14, temp.analog, temp.in);
```

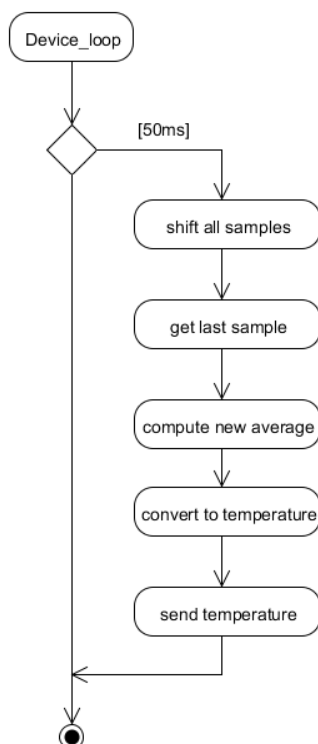
The Pin class is a part of the OpenROV software architecture. It simplifies the connection of new hardware and is able to handle the initialisation of the ADC. This initialisation must then simply be done by the two following commands :

```
1 temp.reset();
2 temp.write(0);
```

Once this initialisation done, the value of the ADC can be read using the method `analogRead(14)` of the Arduino library.

It must be noted that the ATMEGA2560 embedded on the OpenROV Controller board, has a 10-bit ADC that do the conversion for a range from 0 to 5 volts. Which means that the voltage measured must be converted from the ADC value with:  $V_{in} = ADC_{value} * \frac{5}{1023} \approx 0.0049$

### 7.7.2 Noise filter implementation



Once the value is retrieved from the OpenROVController Board ADC, the filter described in section 7.6 must be implemented. The implementation is entirely done in the `Device_loop` and is shown in the figure 49. Samples are taken every 50ms using the Timer `temperaturesample`, initiated in the `Device_setup`. All samples are stored into a 50 elements array, and are shifted before the last sample is obtained from the ADC. A new average is then computed. This averaged value can then be converted into a temperature measurement and be sent to the BeagleBone Black.

Figure 49: Filter's UML flow diagram

### 7.7.3 Voltage to temperature conversion

The value obtained after the ADC is not the thermistor's resistance but a value based on the output voltage of the circuit described in 7.5. It is necessary to compute the corresponding resistance. First, the voltage needs

to be recuperated from the ADC value, which is done by multiplying the ADC value by 0.0049 (see section 7.7.1).

The circuit being composed of a voltage divider, whose output is connected to an amplifier, the gain G of the amplifier needs first to be removed.

$$V = V_{out}/G \quad (4)$$

with G measured on the circuit at 2.486.

This gives the voltage at the output of the voltage divider. The Resistance of the thermistor in the voltage divider can then be computed :

$$R_{thermistor} = \frac{V * R_{resistance}}{V_{in} - V} \quad (5)$$

$V_{in}$  is 4.096, defined by the voltage regulator.

$R_{resistance}$  is the second resistor in the divider, which is 47k.

Once computed, the resistance of the thermistor must be converted into a corresponding temperature, which can be done by two different ways :

The Most precise one is to use a lookup table. For each thermistor, a lookup table is normally available and can be directly implemented. It is however a very heavy implementation and more generic method are available that would be easier to change in case of a modification of thermistor.

The other solution, which was chosen, is to use the Steinhart Hart equation. This equation allows to know the corresponding temperature for a given resistance, knowing the properties of the thermistor.

This equation is the following :

$$\frac{1}{T} = A + B \ln(R) + C(\ln(R))^3 \quad (6)$$

T is the temperature in kelvin and R is the resistance of the thermistor.

A, B and C depend on the the thermistor, but are rarely given in the thermistor data-sheet. For this reason, it is necessary to use an adaptation of this equation which use the resistance of the thermistor at 25 °C and its  $\beta$  coefficient

The Steinhart Hart equation becomes then :

$$T2 = (T1 * \beta) / \ln \frac{R25}{R} / (\frac{\beta}{\ln \frac{R25}{R}} - T1); \quad (7)$$

T2 is the temperature

T1 is the ambient temperature in kelvin, which is 298.15 °K

R25 is the resistance of the thermistor at 25 °C which is generally given in the thermistor data-sheet. It is 10k for the chosen thermistor.

$\beta$  is a coefficient generally given in the thermistor data-sheet, which is 3892 for the chosen thermistor.

R is the current resistance of the thermistor.

All the calculation above are done in the function `float get_temp(float)`, which is called after filtering in the `Device_loop`.

## 7.8 BeagleBone Black and browser software

The Node.js Code concerning the temperature measurement is in the `index.js` of the temperature plugin.

This code only forwards the temperature values in the globalEvent loop for logging, in the same way that it was done with the conductivity plugin. (see section 6.8.2)

No code for the browser was written for this plug-in. However, an interface allowing to calibrate the temperature sensor should be implemented in a future development.

## 7.9 Tests

### 7.9.1 Accuracy

The accuracy of the temperature measurement was tested in two ways.

Firstly, it has been compared to a calibrated Thermocouple type E, built on-site at the Glenn research center, by writing down the values of both sensors obtained in a solution for various temperature. This measurement, shown in the figure 50 below shows a very good accuracy over the entire range of the sensor. Both curves on this figure are almost completely merged.

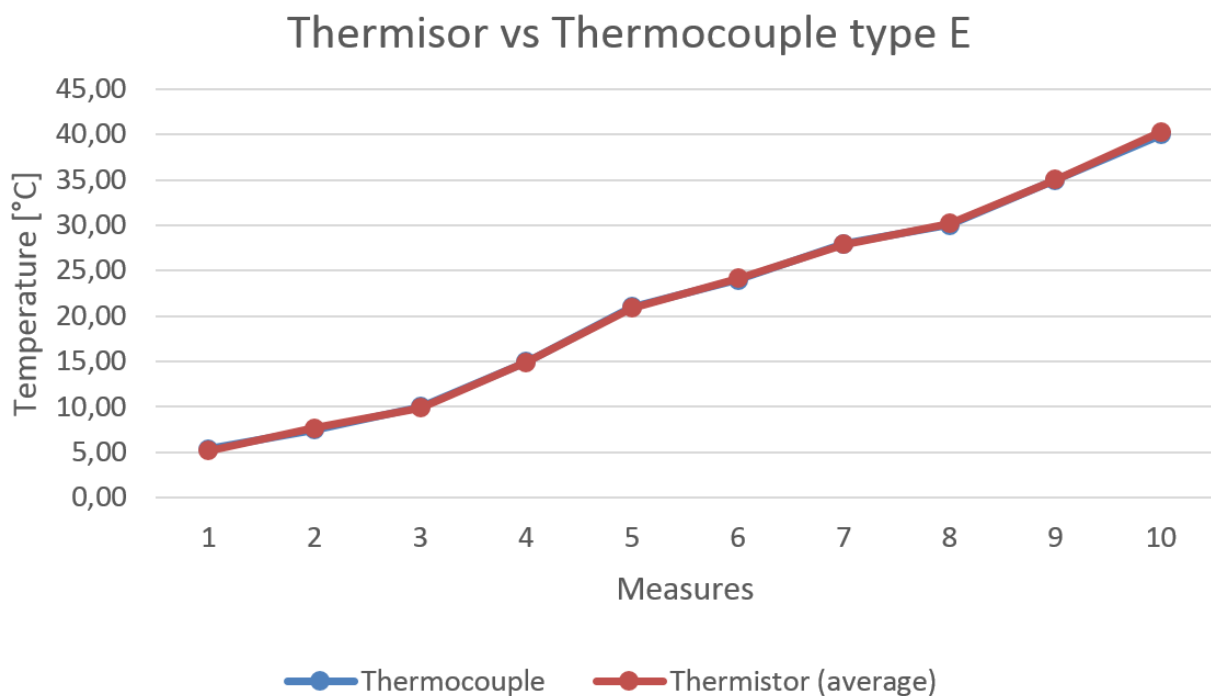


Figure 50: Thermistor vs Thermocouple

However, the values displayed in figure 50 are the average of the measured values. This average has an absolute error of  $\pm 0.1$  °C over the entire range, when compared to the Thermocouple. The values of the thermistor without being averaged had precision of  $\pm 0.6$  °C over the entire range.

It must be considered that the test above was done before the filter was implemented. Another test should be done in the current version in order to validate those results.

The sensor was also tested alongside the same thermocouple to various places in the lake (with the filter implemented), and showed good accuracy as it can be seen in the figure 51 below. All measurement except one had errors of maximum 0.2 °C. One of the measurement has a difference of 0.85 °C, but after further analyses, the temperature at this location dropped by more than 1 °C at 1 meter deep, depth at which the measurement could have been taken.

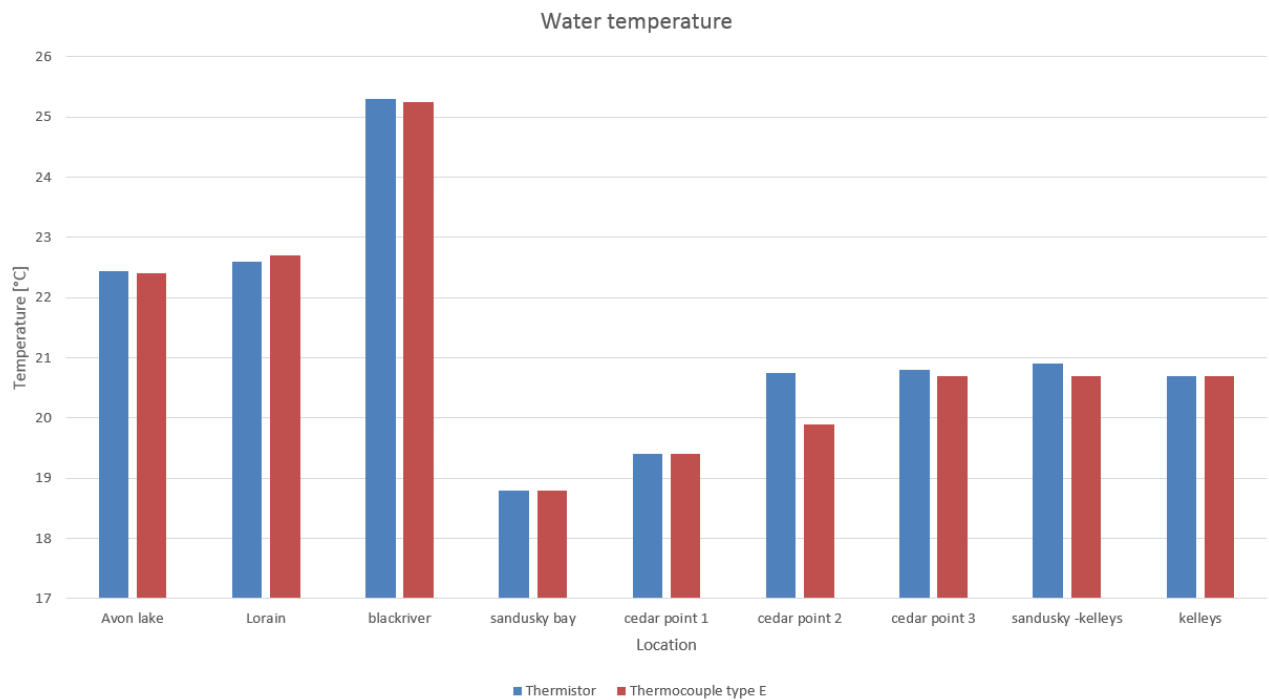


Figure 51: Thermistor vs Thermocouple : field measurements

### 7.9.2 Filter and precision

A test was realised in order to check the efficacy of the sensor. The test consisted in tacking a measurement over a long period of time without the sensor and then do the same test again but with the filter activated.

The results of this test can be seen in the figures 52 and 53 below:



Figure 52: Temperature measurement without the filter

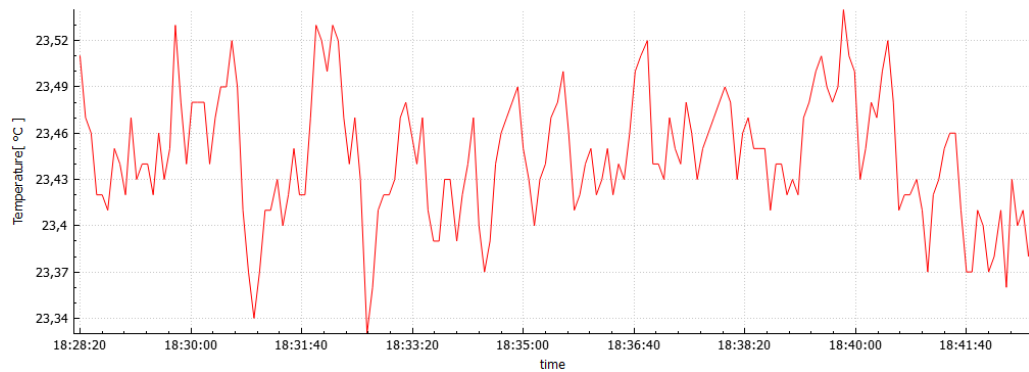


Figure 53: Temperature measurement with the filter

It can be seen in the pictures above that the precision increased from approximately  $\pm 0.3$  °C to  $\pm 0.1$  °C. This test is however not sufficient for a proper evaluation of the sensor's precision, since it was done only at ambient temperature. The sensitivity of the thermistor measurement varies over the range, which makes additional tests even more indispensable. For example, during the accuracy test in section 7.9.1, precision of  $\pm 0.6$  °C was found in the worst cases, which demonstrate the existing difference of sensitivity.

### 7.9.3 Sensibility

The main interest of building a thermometer based on a thermistor was to detect quick change in temperature, which is especially important because the rover is moving. The measurements taken on the lake showed a few examples of situation in which this sensitivity was useful, as in the example showed below between the thermistor and the IMU/Depth temperature sensor. The thermistor indicates at 14:16:30 a temperature of approximately 19.8 °C when the IMU/Depth sensor is still at a higher temperature of 20.1 °C.

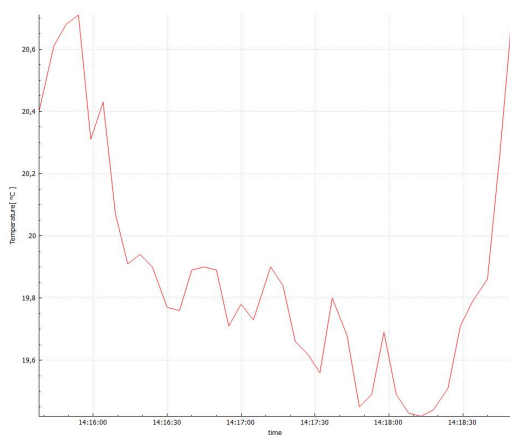


Figure 54: Measure with thermistor

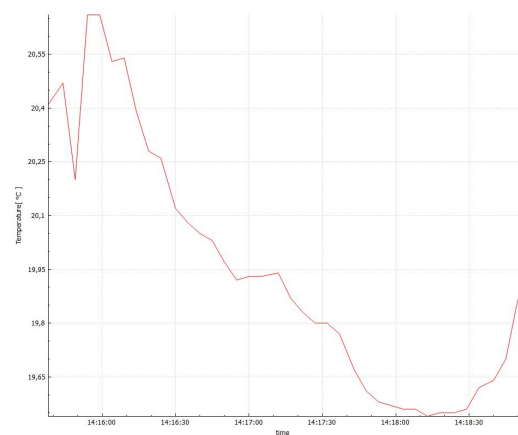


Figure 55: Measure with IMU/Depth Sensor

The advantage of the thermistor is even more visible in a scatter plot where the temperature is plotted in function of depth as in the figures 57 and 56. we can see an hysteresis that is not here with the thermistor but is visible the IMU/Depth sensor, due to its low thermal time constant.



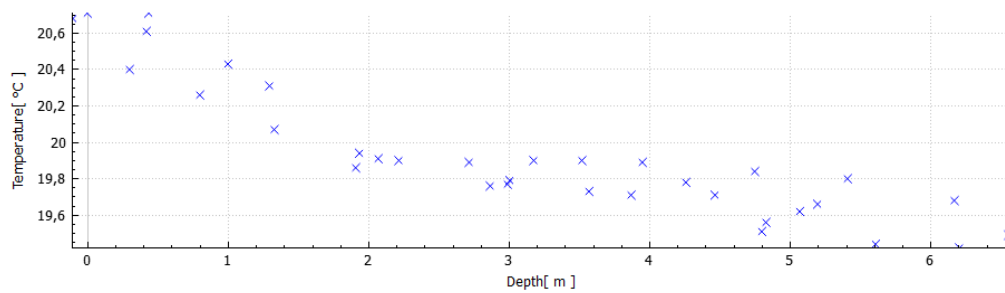


Figure 56: Scatter plot with thermistor

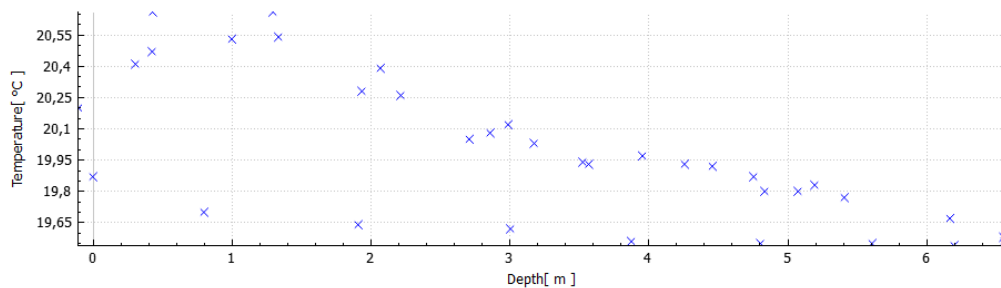


Figure 57: Scatter plot with IMU/Depth Sensor

## 7.10 Calibration

No calibration procedure was implemented. One of the advantages of the thermistor is that they can be found with a very high accuracy. The selected thermistor has an accuracy of 0.1 °C and can then easily be changed without degrading the results. However, after some time some components on the board might get older and slightly changed which could slightly change the precision of the sensor. In this case some change might be needed in the C++ code to correct the temperature value.

## 7.11 Conclusion

The temperature measurement was developed to an exploitable point. The accuracy could be tested through the entire range with good results. The Steinhart Hart equation gave for this matter an unexpectedly good result in order to retrieve the temperature from the resistance measurement. The precision however was only tested for an ambient temperature, further tests are therefore necessary in order to properly verify it over the entire range.

The main objective of this measurement, which is to take quick measurement of the temperature could be accomplished with success. The circuit driving the thermistor had however to be developed quickly and ameliorations of this circuit could considerably improve the measurement's precision.

There is still a feature that should be implemented: no simplified calibration procedure was developed for the user, and the only way to do it is to take measurements on the temperature board and modify the coefficients in the temperature.cpp file. An interface could be implemented as a browser plugin as it is for the conductivity sensor in order to simplify the adaptation of those coefficients.

## 8 RGB SENSOR

### 8.1 Introduction

#### 8.1.1 Turbidity and transparency

Turbidity is a measure of the cloudiness of water and how suspended solids affect the passage of light through water, more specifically how the light is scattered by materials in the water. There are several ways to measure turbidity: Measuring the attenuation of light as it passes through the water column is the most direct way, It is also very common to measure directly how the light is scattered through the water. A very simple and cheap way is to use a Secchi disk, which is a black and white disk that is lowered into the water until it disappears, this depth being recorded as a measure of the water transparency which is inversely related to turbidity.

The measure of Turbidity is closely related to the measure of TSS (Total suspended solids) which is measured in [mg/l]. Suspended solids are typically in size range of 0.004 mm to 1.0 mm[1]. The composition of suspended solids is however different for every environment and can be only *estimated* in quantity from a turbidity measurement. A high turbidity can be due to various materials such as sediments, inorganic and organic matter, waste discharge, microscopic organisms and more specifically for the study in lake Erie : algae.

High turbidity can affect other water quality as well. The temperature is likely to be higher in turbid environment as the suspended particles absorb more heat which in turn reduces the concentration in dissolved oxygen (DO) because warm water holds less DO than cold [1] [5]. Higher turbidity also reduces the amount of light penetrating the water, which reduces photosynthesis and the production of DO[1]. However, this might not be always the case. Since the turbidity is likely to be due to algae in lake Erie, the photosynthesis resulting of this proliferation is likely to compensate this. Particles also provide attachment places for other pollutants, notably metals and bacteria[6].

#### 8.1.2 Visible light and algae detection

Light attenuation in the visible spectrum can be used to measure the presence of algae. This being because of the light absorption of the chlorophyll, which is present in algae. Two type of chlorophyll can be present : chlorophyll a and b, both having a different electromagnetic absorption curve as it can be seen in the figure 58.

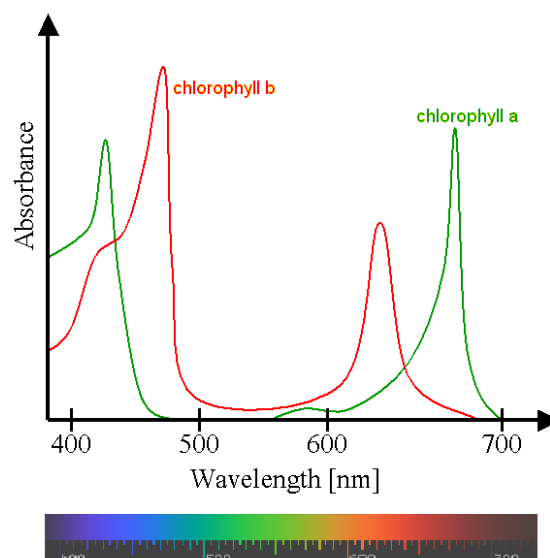


Figure 58: Chlorophyll light absorption spectrum

It can be seen on this picture than both sorts of chlorophyll will absorb different part of the electromagnetic spectrum but always in the visible spectrum corresponding to red and blue light and never the green light.

This property however can't be directly used by measuring the composition of light under water without taking other factor in consideration. The first parameter to consider is the composition of the light emitted by the sun, which is not equal for all wavelength. As it can be seen in the figure 59 below, the lower wavelength are more present.

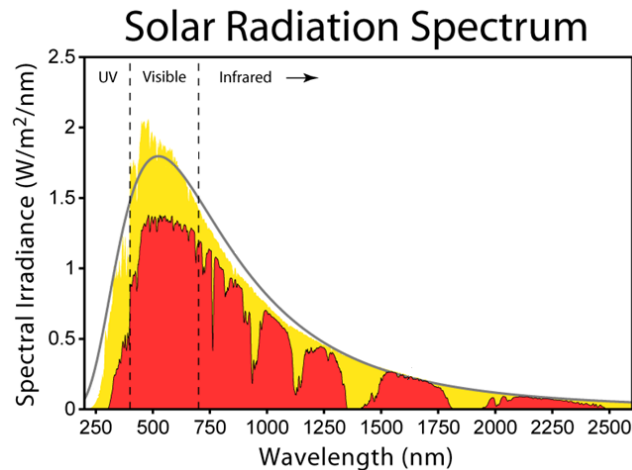


Figure 59: Sun's light radiation spectrum

This means that the measurements of the electromagnetic spectrum absorption in medium must always be relative to the ambient light at the surface.

Another point to take into consideration is the absorption of the electromagnetic spectrum into water. Even if the water is pure, the light will not be absorbed equally as it can be seen in the picture below.

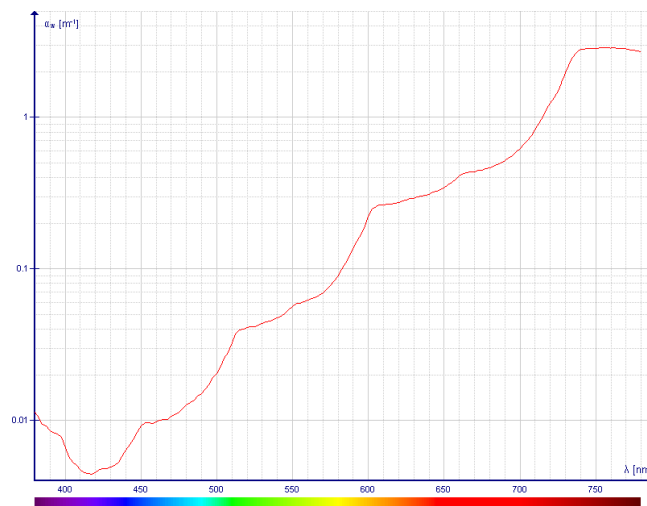


Figure 60: Absorption of light into pure water

The red light will be absorbed more quickly than the blue and green, which should not be a surprise when data are analysed.

Taking all the points described above, it can be seen than the presence of chlorophyll will especially change the results between the blue and the green spectrum. The blue spectrum will be greatly absorbed by the chlorophyll, letting mostly the green spectrum go through the water column. It must however be taken into consideration than other elements into water could also cause this kind of behaviour. But despite this, changes in those absorptions ratio that can be observed by constant monitoring throughout the year could give precious information.

### 8.1.3 Sensor's choice

The two water quality parameters described above are normally measured by very expensive sensors. Being too expensive, alternatives had to be found. Cheaper sensors can be found that measure the light and can be used to monitor how the amount of light decreases in the water column. This allows to obtain a measurement of the transparency of the water and can substitute to a turbidity meter.

Moreover, some of those sensors are able to measure light at different wavelengths in order to output the colour of the water. Measuring separately those different channels allows to observe differences into their respective absorption and thus enable to determine the presence of different elements into water, such as chlorophyll.

## 8.2 Architecture

The architecture of Colour/Light measurement is shown in the figure 61 below:

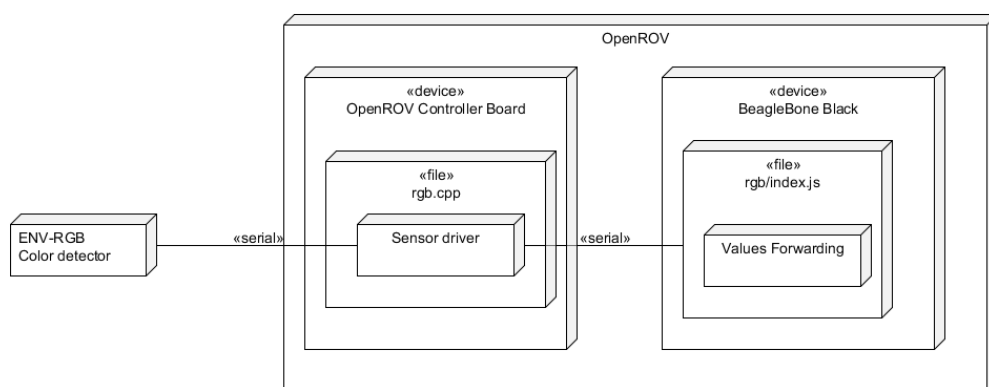


Figure 61: UML physical diagram of the colour/light measurement

The sensor is directly connected to the OpenROV Controller board and driven in the file `rgb.cpp`, which makes sure that the sensor is working in the proper state and forwards the gathered values that are sent by the sensor every 1.2 seconds. Those values are sent to the plugin `rgb` on the BeagleBone Black, which simply forwards the values to the global event loop.

A block diagram of this architecture is shown in the figure 62 below.

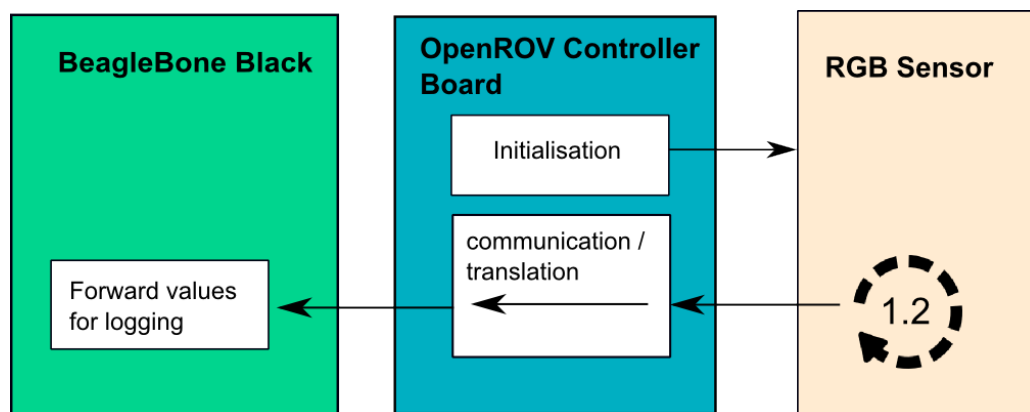


Figure 62: Software architecture of the conductivity measurement

## 8.3 Sensor

The sensor chosen to measure the parameters explained in section 8.1.2 and 8.1.1 is the ENV-RGB from Atlas Scientific. The reason for this choice is essentially its very low cost, which is one of the main aim for this

project. For 48\$, this sensor is able to give multiple indication of water quality. The main aim of this sensor is to output the colour of the water, but in order to do so, the values effectively measured are the light received in different parts of the spectrum. Those measurement of light being also available it is then possible to extend the use of the sensor for a measure of the water transparency by measuring the absorption of light through the water column as explained in section 8.1.1. Moreover, light in the the red, green and blue spectrum being measured separately, it is also possible to analyse the difference of absorption between them in order to detect element in the water as explained in section 8.1.2. Further explanations about how this sensor can be used for the measures of those parameters will be explained in the following sections.

### 8.3.1 Measure of turbidity

Turbidity, as explained in the section 8.1.1 can be measured by the light absorption and scattering throughout the water column. This sensor being able to output the total light, it is possible to analyse how this light diminished in function of depth and thus to know the transparency of the water.

This sensor has however a major drawback for this analysis, its saturation is very low (3'200 lx) and is not able to measure properly the light outside of the water. This problem can be avoided by different means. The best one would be to use an attenuator calibrated for the current conditions before a measure. This solution however has not been tested by lack of time. The second solution, which has was chosen is to change the positioning of the sensor to limit the amount of light reaching it. This solution is explained in the section 8.5

### 8.3.2 Detection of algae

The presence of chlorophyll can be detected using this sensor, however, this detection is more of an approximation since this sensor is not able to analyse the full visible spectrum at specific wavelength. Very few informations are given in the sensor data-sheet about the exact sensibility of the sensor throughout the electromagnetic spectrum. The only information given is that the sensibility is higher at 640nm, 524nm and 470nm wavelength. For this reason it is very difficult to know how the light detected by the sensor will exactly change in the presence of chlorophyll without experimentation. What can be known however, is that if the blue light is absorbed more than the green light throughout the water column, the presence of chlorophyll is very likely as explained in the section 8.1.2.

It will unfortunately be impossible to know if the chlorophyll a or b is more present since the highest sensitivity in the blue spectrum is indicated to be 640nm for the sensor, wavelength for which the two type of chlorophyll have a similar absorption as it can be seen in the figure 58. The sensor is more sensitive in the red part of the spectrum at 470nm which would then be more attenuated by the chlorophyll b, but the red light is more difficult to analyse since it is also absorbed strongly by the water itself as shown in the figure 60.

Other sensors exist that will give a precise measure of chlorophyll using essentially fluorimetry but those are very expensive and were not chosen for this reason.

## 8.4 Connections

The RGB Sensor must be connected to the OpenROV Controller Board through 4 wires. 2 wires are used to power the sensor and the 2 other wires are used for the serial communication with the OpenROV. The probe is waterproof, certified IP68.

The figure 63 shows the different connections between the sensor and the OpenROV Controller board, including to which Open- ROV Controller board I/O the wires from the RGB Sensor are connected :

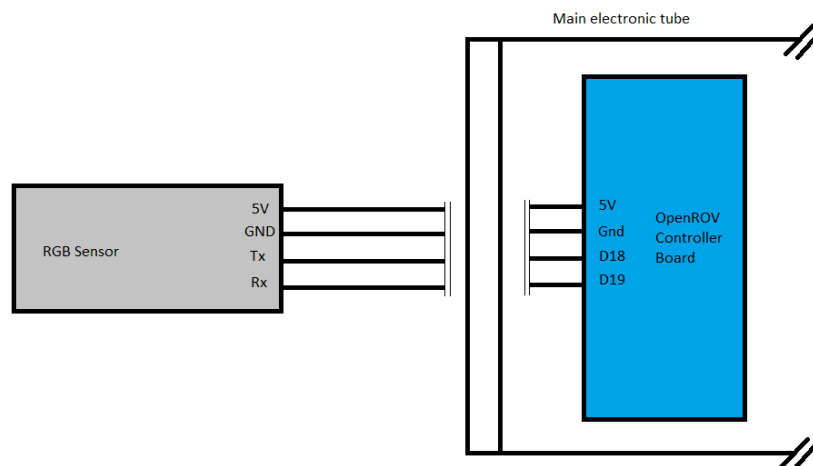


Figure 63: Connections for the RGB Sensor

## 8.5 Sensor positioning

The ideal positioning of the sensor is to look directly above the rover, in direction of the surface, which would allow it to look at the light outside of the water when the rover is at the surface and then at different depth when lowered into the water column.

However, as explained in the section 8.3.1, the light reaching the sensor is too high during sunny days. For this reason, It was chosen to place the sensor looking downward into water as shown in the figure 64 below:

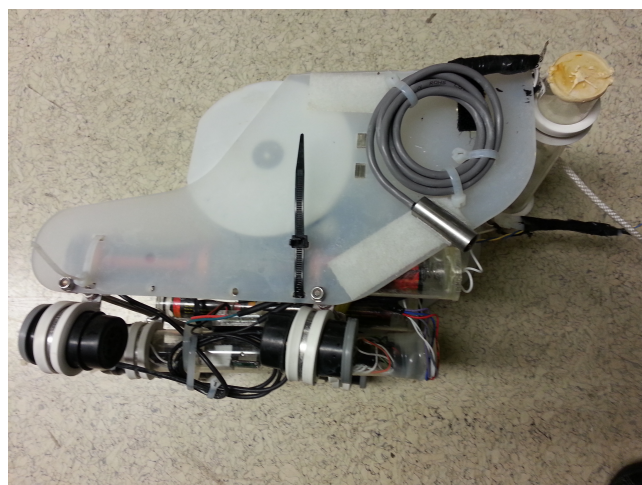


Figure 64: Optimal positioning of the sensor

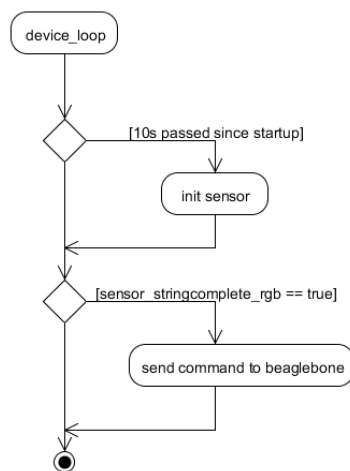
This has several drawback :

First, the light outside of the water is not measured and thus, the light that might be reflected at the surface and/or absorbed the first centimetres is not taken into considerations.

Secondly, the floor can have an influence on the measurement when close to the sensor. This means that the measures close to the bottom of the lake must be discarded.

## 8.6 OpenROV Controller board software

The driver code developed in the C++ plugin, consist in the initialisation of the the sensor at the desired default state, the communication with the sensor and the forwarding of the values to the BeagleBone. It is very similar that what has been done with the Conductivity sensor in section 6 but in a simpler version. Since no calibration is necessary for this sensor and the interest of sending commands to this sensor is limited, the communication has only been implemented to send messages from the OpenROV Controller board to the BeagleBone.



The design of the plugin code can be seen in the figure 65. A timer is used at startup to send two commands to the sensor which are :

- ◆ M3 that ask the sensor to send all the values of colour and light available
- ◆ C that ask the sensor to send the values every 1200ms which is the minimum time possible

Then, the program will constantly receive commands from the sensor, in the exact same way than the conductivity sensor (see section 6.9.2)

Figure 65: UML flow diagram for rgb.cpp

## 8.7 BeagleBone and Browser Code

In the BeagleBone, the values received by the sensor are simply forwarded through a global event loop event in order to be logged into the database by the dedicated plugin. This is done in the same way that it was done with the conductivity plugin. (see section 6.8.2).

No code was written from the browser and none is necessary for the proper functioning of this sensor.

## 8.8 Calibration

No calibration is necessary nor available for this sensor. 5 commands can be used to control the sensor, which only allow to request output or choose the kind of output that is desired. Those options should not be modifiable by the user as it would impact the proper functioning of the logging system.

## 8.9 Tests

Unfortunately, few tests were done with this sensor.

Something that should be done is to compare the data returned by the sensor for the total amount of light with a calibrated light sensor. Unfortunately, no other sensors were available for this comparison during the project. This test is specifically important to check that the results concerning the transparency are relevant when using this sensor, especially since it is not its primary use.

## 8.10 Conclusion

The development made for this sensor concerned only the basic features necessary to operate it and get measurements, which could be accomplished successfully.

However new features could be implemented in the future to help for the monitoring. The analysis of data gathered with this sensor could be made easier by processing the sensor's values into more comprehensible data. For example, the ratios between the different colour channels could be compared and a value that corresponds directly to the possible presence of chlorophyll could be sent.

The positioning of the sensor is also not optimal in the current setup (see section 8.5), which could be ameliorated in future developments.



## 9 DATABASE

### 9.1 Database system choice

Data storage is a major part of the project: Measurement taken by the sensors must be stored somehow, in a way that allows to handle them in a convenient way in order to be able to get practical informations easily. An SQL database is an interesting choice in this case: data can be well organized and accessed through SQL requests that allows to access easily specific datasets and pre-process them. Using simple text file such as XML or CSV would be more difficult to handle and less versatile in various situations. It is also simple to parse those files from an SQL database if needed for future extensions of the project.

Among the different database systems available, most of them are over sized for a simple data-logging use and are difficult to implement and administrate. The measure system only needs to be able to store data in an organized manner and access them through SQL. Advanced SQL system proposing various features such as access control, high concurrency or high storage capability are not needed.

The best fitting system seems to be SQLite. SQLite is a very small system, which is interesting to be embedded in a system such as the OpenROV. It is also very simple to use. SQLite has some limitations comparing to other larger systems, however, most of those are not a problem in this kind of logging use. The only possibly restricting limitation is the fact that only one client is able to write the file at a time. However, This is not a problem in this project since the file is written only by the logging plugin and is exported for external uses.

### 9.2 Sqlite3 installation

SQLite3 needs to be installed on the BeagleBone Black and will be accessed through the node.js program. One way to install it is to download the source code archive `sqlite-autoconf-3080500.tar.gz` and build it on the BeagleBone through the following command sequence :

```
1
2 ./configure --prefix=/usr/local
3 \make
4 \make install
```

However, an error might occurs when running `sqlite3:SQLite header and source version mismatch`. In case this error appears, the file `/usr/lib/x86_64-linux-gnu/libsqlite3.so.0.8.6` needs to be replaced by the the version present in `/usr/lib (/usr/lib/i386-linux-gnu/libsqlite3.so.0.8.6)`

The access to `sqlite3` is made through the `node-sqlite3` module for `node.js`. This module can be installed by launching the following command :

```
1 npm install sqlite3
```

For organization purpose it is good to have all `node.js` modules installed in the same place in `/opt/openrov/node_modules`. To do so, the command should be launched from `/opt/openrov`

### 9.3 Database architecture

The UML schema for the database is the following :

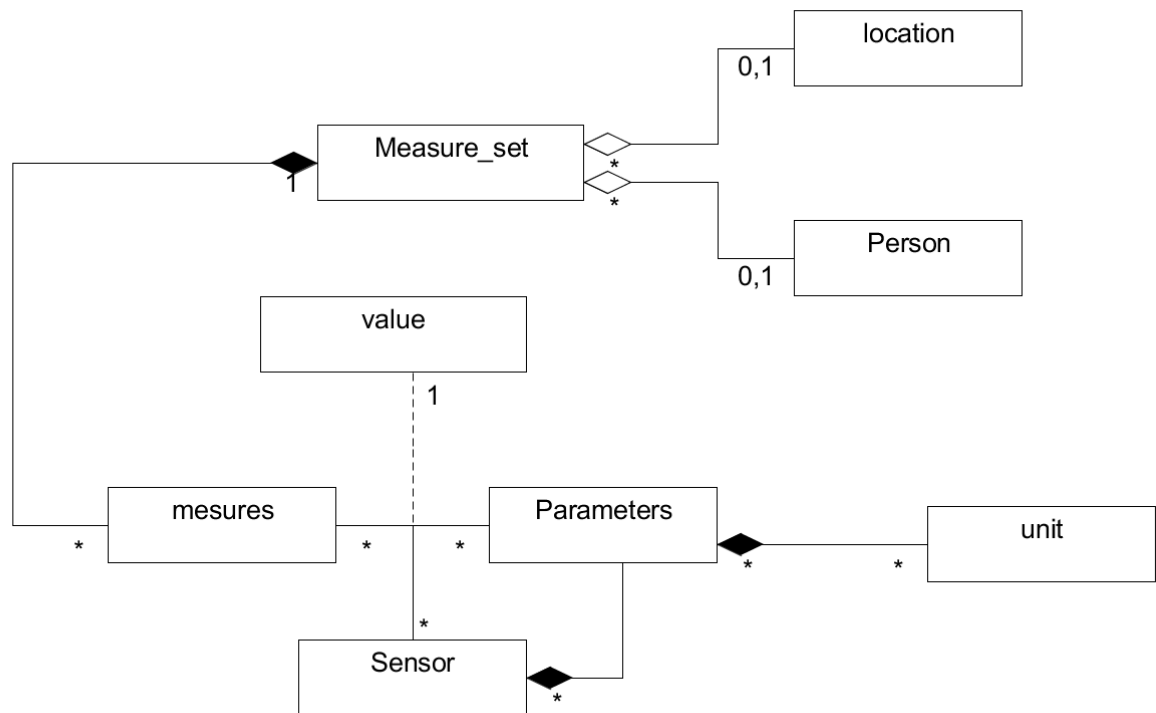


Figure 66: Database UML diagram

The table `MeasureSet` corresponds to a measurement launched from the rover, which corresponds in fact to a group of real measurements, taken at a regular interval. If the user wishes so, he can link this measurement to a location and/or an operator.

Those real measurements can be inserted into the table `Measures` which represent the time at which a measurement was taken. each sample being linked to several couples parameters/sensors to which a value is assigned. By doing so, each measured value is linked to the time it was taken, which sensor took it and what is the parameter this value represent.

Each parameter needs then to be connected with at least one unit. Several units could be used such as feet instead of meters, this is the reason the relation between the parameters and units is many to many.

An other link was made between the Sensors and the parameters. If this wasn't done, no natural way to get the relations between the sensors and the parameters they measure would be available (passing through the table `measures` would make no sense and wouldn't be always possible).

For a better understanding of this architecture, it can be seen as 2 independent sides : One describes the capabilities of the system (which sensors are embedded and which parameters the system is able to measure) and the other describes a point from which measurements were taken. Those 2 sides are linked when measurements are taken through the table `Measures` and the values linked to each parameters.

The full relational diagram of this database is shown in annexe J. 3 tables that implements all relations many to many were added to obtain this structure. Those relations are :

- ◆ The parameters to the units in table `Units_Parameters`
- ◆ The values linked to measures, parameters and sensors in table `Measures_Parameters_Sensors`
- ◆ Parameters to the sensors in table `Parameters_Sensors`

## 10 LOGGING

### 10.1 Architecture

The logging system is entirely written in the plugin named `sensors` on the BeagleBone Black. A block diagram of its architecture can be seen in the figure 67 below :

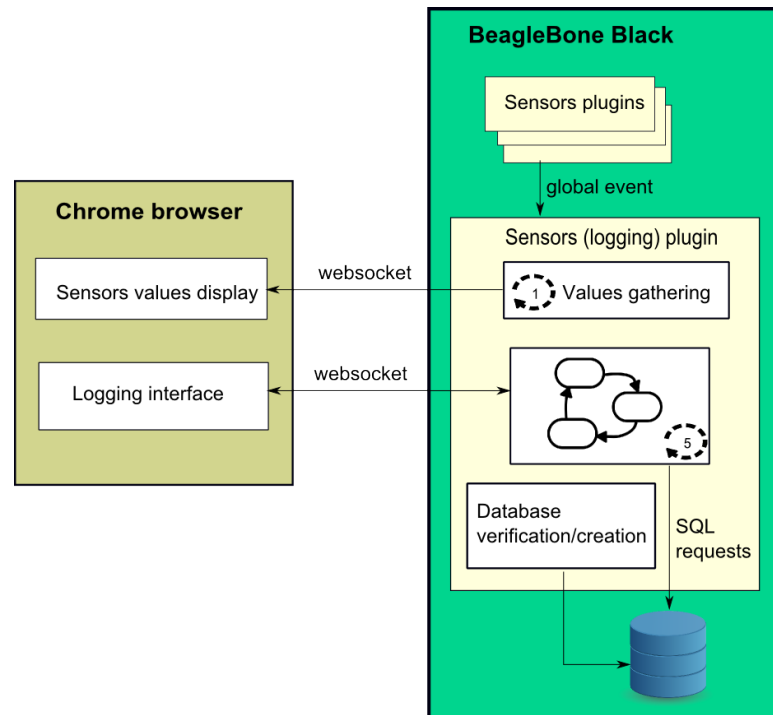


Figure 67: Architecture for the logging

All the logging system is built around a state machine driven from the browser. The user can decide to take a measurement, which allows the state machine to pass in a logging state in which data from the sensors are periodically recorded. Those data are recorded into the database described in section 9.

A few elements in the `index.js` file are however independent from this state machine. Those elements are :

- ◆ The retrieval of the sensor's values, that are regularly sent to the browser for display through a websocket.
- ◆ The creation of the database at startup if it doesn't exist.

In addition to the usual Node.js and browser files present in the BeagleBone's plugins, a few other files are necessary in order to create the database:

- ◆ A `plugins/sensors/database.sql` file that contains the script necessary for the database creation.
- ◆ A `plugins/sensors/base_dataset.sql` file that contains entries in the database that describe the system (parameters recorded, sensors embedded, ...)

The database file itself is also stored in the plugin folder in `plugins/sensors/measure.sql`. It is the file that needs to be downloaded in order to analyse the measures.

## 10.2 Data reception

Every value that can be logged into the database are sent through the global event loop from each sensor's `index.js` file.

However, in order to be able to log properly the data when a sensor is added, a few element must be modified into the system before.

Information about the sensors and their corresponding parameters must be added to the database. This is done by modifying the file `base_dataset.sql`. The exemple below shows the necessary lines written for the temperature sensors :

The parameter and its corresponding units must be added first and linked together :

```
1 INSERT INTO Units VALUES (1, "degres Celcius", "C", "1");
2 INSERT INTO Units VALUES (2, "degres Farenheit", "F", "x * (9/5) + 32");
3
4
5 INSERT INTO Parameters VALUES (2, "Temperature");
6
7 INSERT INTO Parameters_Units VALUES (2,1);
8 INSERT INTO Parameters_Units VALUES (2,2);
```

The sensors that measures the entered parameters can be added and also linked with their corresponding parameters :

```
1 INSERT INTO Sensors VALUES (3, "Thermistor", " - ", "driven by home made circuit");
2 INSERT INTO Sensors VALUES (2, "IMU/Depth", "OPENROV", "bought with OpenROV");
3
4
5 INSERT INTO Parameters_Sensors VALUES (2,3);
6 INSERT INTO Parameters_Sensors VALUES (2,2);
```

Once this is done, the `index.js` file must be modified to log the new parameters.

First, a structure called `sensorsvalues` defined in this file handles the link between the values to the corresponding sensor and parameters in the database and must be modified to reflect the database entry added previously. This structure is used by the logging system in order to write into the database and can be seen below :

```
1 var sensorsvalues = {
2   watertemperature: {value: null, sensor: 3, parameter: 2},
3   imutemperature: {value: null, sensor: 2, parameter: 2},
4   depth: { value: null, sensor: 2, parameter: 6},
5 };
```

The values indicated into the `sensor` and `parameter` fields are their corresponding ID in the database.

Once this is done, the values can be retrieved from the global event loop and the values copied into the structure as in the exemple below (still in the `index.js` file)

```
1 deps.globalEventLoop.on('temperature', function(data) {
2   if('temp' in data) {
3     sensorsvalues.watertemperature.value = data.temp;
4   }
5 });
```

## 10.3 Database creation

The presence of the database is checked every time the system is booted up. If no database is present, it is created according to the `database.sql` file.

the creation of the database is done in the function `databasecreate()` in the `index.js` file

## 10.4 User view

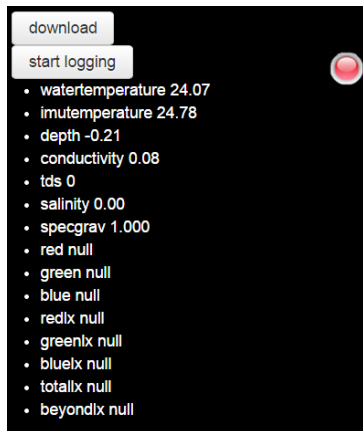


Figure 68: Interface for the sensors plugin

The main user view corresponding to the `sensors` plugin can be seen in figure 68. This interface is visible on the cockpit interface, on the right side.

The user can launch or stop the logging by using the button `start|stop logging` (The button's label changes as well as the light's colour depending of the logging state). The current values of all sensors are also displayed, and are regularly received through a websocket.

The user can also download the database containing all measures taken.

When the button `start|stop` is clicked, a tab is opened on the right side of the cockpit interface, in which a form (figure 69) can be filled with the informations necessary to launch the measurement. Only the name of the measurement must be filled. The other information that can be entered such as the Operator or the location are optional. If an Operator or the location

need to be added, the form need to change. The new form that is displayed is received through the websocket message `changeform(html)` from the statemachine.

### New measure point

name:

Operator: Yannick Baehler

location:

- kelleys island
- kelleys-sandusky
- cedar point 3
- cedar point 2
- cedar point 1
- Sandusky Station 4
- Sandusky Station 3
- Sandusky Station 2 (KSU Stn 20)**
- Sandusky Station 1 (KSU stn 19)
- Sandusky channel bells
- lab

Figure 69: Logging form

When those forms are validated, the information they contain is sent to the state machine through websockets.

## 10.5 State Machine

The logging system is managed through the following state machine :

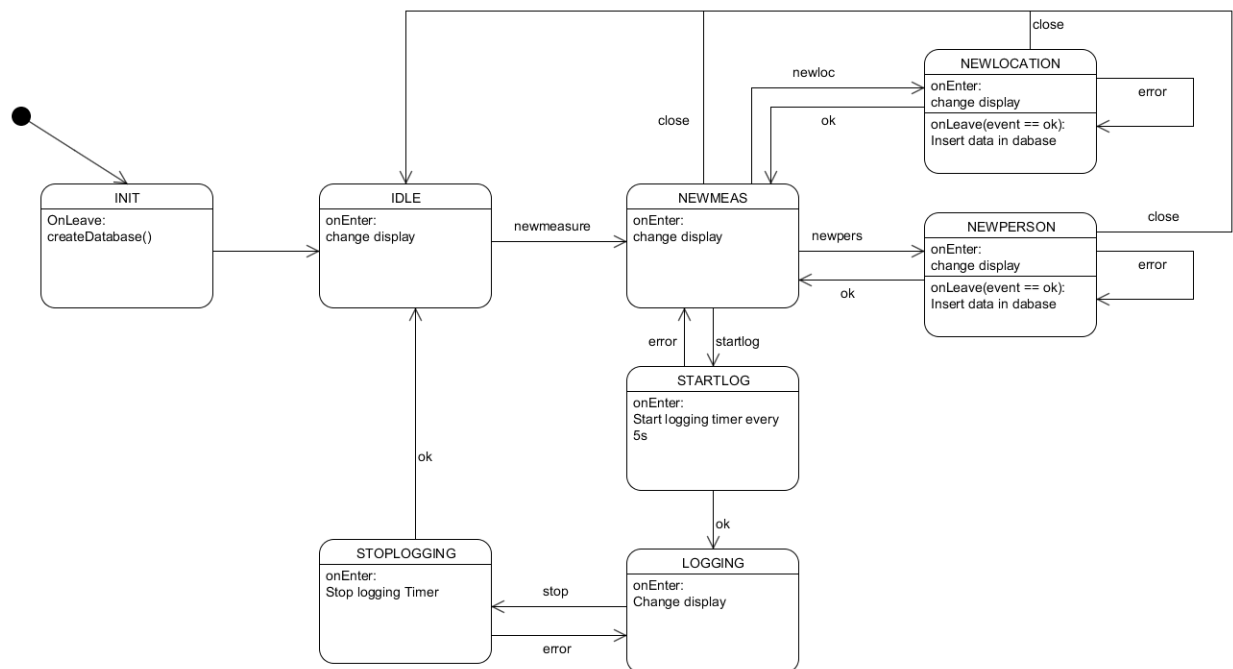


Figure 70: Login state machine

This state machine is entirely driven from the browser and can mostly be understood by looking at the user interface as shown in section 10.4.

When the user wishes to start a new measurement, the state machine enters in the `NEWMEAS` state. In this state it can either choose to add information about the measure, or add information about the operator (State `NEWPERSON`) or the location (State `NEWLOCATION`).

Once the user confirms that the system must start to log data. The logging is initialized in the State `STARTLOG` and the state machine will then automatically enters in the state `LOGGING`, where it would stay until the user wishes to end the measurement. In this state, data are logged every 5 seconds into the database.

Once the user wishes to stop the measurement, the state machine will enter in the state `STOPLOGGING` and automatically come back in the `IDLE` state, ready for the next measurement.

The events driving the state machine are mostly coming from the browser interface and are shown in the figure 71:

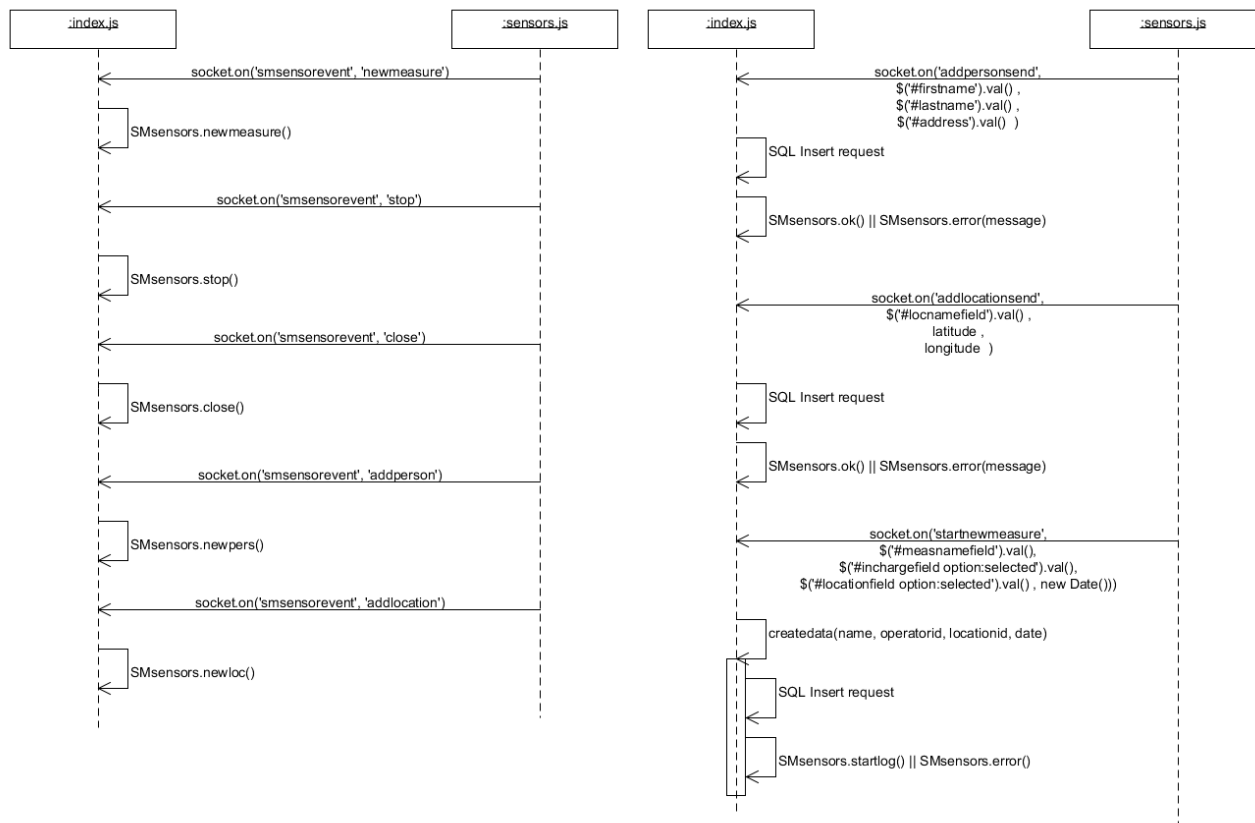


Figure 71: Logging state machine: Events from browser

The event shown in the right side of the figure 71 correspond to the validation of the forms before a measurement is taken. The information contained in the forms is tried to be registered into the database and the events resulting depend on the successes of this insertion.

The only events that have not an origin from the browser are the events generated in the states `STARTLOG` and `STOPLOGGING`, which are transitory states.

## 10.6 Logging

The logging of the data is launched by the user from the browser interface and a websocket signal `startnewmeasure` is sent to the BeagleBone Black, containing the informations concerning the datapoint selected by the user and the time on the user's device. The logging is done as shown in the figure 72.

When the websocket signal is received, the function `createdata(name, operatorid, locationid, date)` is called, in which an entry is created in the database for the measurepoint. If the entry can be added into the database, an event `startlog()` is sent to the state machine, starting the logging.

A timer handling the logging is then launched and data are written into the database at each tick until the measurement is stopped.

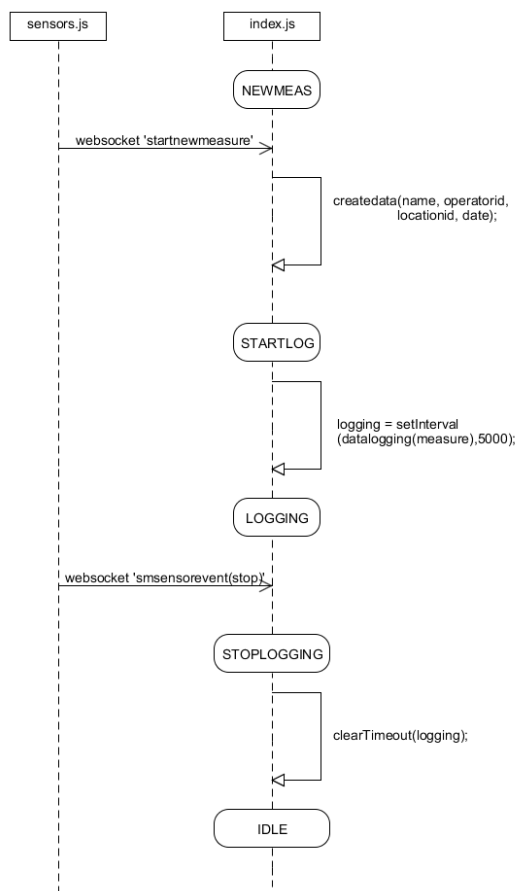


Figure 72: Beginning and end of the logging

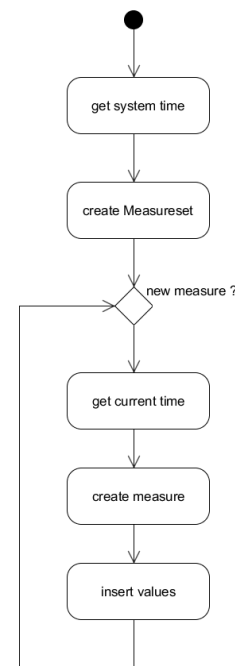


Figure 73: logging activity

The figure 73 above shows the continuity of the logging.

When launching a measurement the SQL request `INSERT INTO MeasureSet VALUES (null,name,date,operatorid,locationid);` is sent (all parameters are received from the browser). A SQL request is then executed afterwards to test that the data are properly inserted before continuing and starting the timer.

Each time the Timer ticks, the current state of all sensors is registered, this is done by first creating a new measure into the database, linked to the current measureset by sending the SQL request `INSERT INTO Measures VALUES (null,date,currentMeasure);`.

Once this is done, each parameters and sensors must be linked to this measure. This is done by iterating through the structure `sensorsvalues` that contains all links between sensors,parameters and values. The SQL request `INSERT INTO Measures_Parameters_Sensors VALUES (measureid, parameter, sensor, value);` is sent for each entry of this structure.

## 10.7 Future development

The database must be downloaded onto the user's computer in order to analyse the data. However, no functions were implemented allowing to delete this database on the rover and the user is currently forced to do it manually by deleting the database file (`measures.db`) from the cloud9 interface (a new database is created in this case during the next reboot of the rover). An interface on the browser should be developed for this feature.

Another problem that should be solved is the parallelization of the logging. In the version developed, the entries of all parameters into the SQLite database are serialized. This cause sometimes the rover to block for a second, which can sometime be an annoyance.



The solution to this problem, which couldn't be tested, is to modify the loop in which all the parameter are logged by setting a timeout for each parameter.

```
1  for (var i in sensorsvalues) { //loop
2
3      (function(i_intern, id_intern) {
4          setTimeout(
5              (function(obj, measure_id) {
6                  return function() {
7                      //logging here for the parameter i(= obj) for the measure with the id given
8                  }
9              })(i_intern, id_intern))
10         ,loop * 100
11     );
12 } (i, id);
```

All timeouts will launch a callback for each parameter, which will ensure the parallelism of the logging and avoid blocking the Node.js loop.

## 11 ROVER MODIFICATIONS

### 11.1 Introduction

Several modifications were necessary in order to be able to install the sensors on the rover. The problems that caused the necessary modifications were :

- ◆ Sensors need to be connected to the OpenROVController board inside the main electronic tube, a way in must be considered
- ◆ The buoyancy of the rover will change when a sensor is added as it becomes heavier, it is then necessary to be able to correct this change easily
- ◆ Extra sensors can need dedicated electronic circuits, which must be waterproofed
- ◆ Sensor must be attached to the rover

The material used for all the modifications done to the rover as described in the sections below can be found in annexe I.4

### 11.2 Main electronic tube modifications

Modifications need to be done to the main electronic tube to be able to connect extra sensor to the OpenROV-Controller board. To this end, a bunch of wires was simply sealed into one of the endcap and can be used for further installations. Molex connectors were added to the bare wires in order to connect them easily as shown in the picture below :

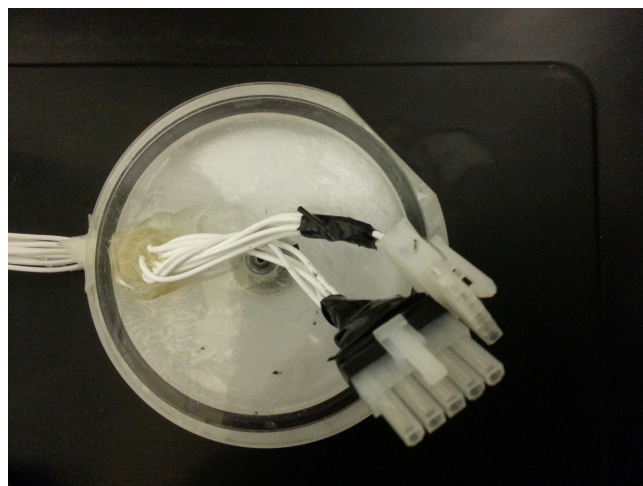


Figure 74: Endcap with extra wires and molex connectors

When the boards of the rover are placed into the main tube, the molex connector can be connected before closing the main tube entirely. The connectors connected to those shown in figure 74 are linked to wires connected directly to the I/Os of the OpenROV Controller board, as shown in the figure 75 below :

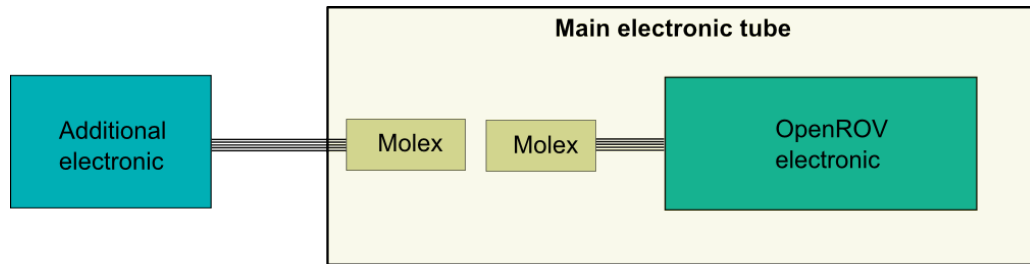


Figure 75: Schema of the connexion through Molex connectors

The potting compound used to seal the wires and cables has to be adapted to the material. A good potting compound that has been using along this project is underwater epoxy. The epoxy that was used is `devcon underwater 2-ton epoxy`. This compound is very fluid and takes about 30 minutes to harden, which gives enough time to properly infiltrate into all asperities and limit the amount of air bubbles that could deteriorate greatly the pressure resistance.

### 11.3 Buoyancy

Adding sensors to the robot implies that its weight is modified and the Archimedes force can be overwhelmed by the gravity force, which causes the rover to sink. The ROV must be neutrally buoyant, it is therefore important to have a system that can adapt easily the robot to a different sensors set up.

The 2 forces that are implied for the buoyancy of the rover are :

- ◆ The Archimedes force :  $F_a = p * V * g$
- ◆ The force of gravity :  $F_p = m * g$

Those 2 forces have to be equal to make the object neutrally buoyant as it is shown in the figure 76:

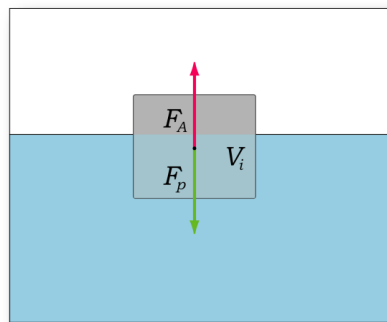


Figure 76: Archimede's law

In order to add sensors to the rover, it is then important to be able to modify the weight for the rover to be equilibrated. A simple way is to use small containers, in which weights can be easily added or removed in case of a modification of the setup.

To do so, 4 containers were added to the rover. It is important to have 4 of them to be able to equilibrate properly the rover if more weight is added on one side of the rover.

In this design the containers used for the weight are 35mm film cases. Those are however not waterproof at high depths and need to be filled with water in order to equilibrate the pressure. To do so, a few holes are drilled into the cases, small enough to avoid the weights to go out.

Small stainless steel balls were used as weights. Since their size is very small, they can be put into a small plastic bags in order to avoid loosing them and keep them packed. Those bags can then be slipped into the containers.

It is necessary to attach the 4 added tubes to the rover. the way that has been chosen to do so is to use tube clamps. Tube clamp are available in materials that are water resistant and anti-slippery. Those tube clamps can be attached using bolts and nuts to the 2 threaded rods of the rover. The needed material, and the resulting assembly can be seen in the pictures below :

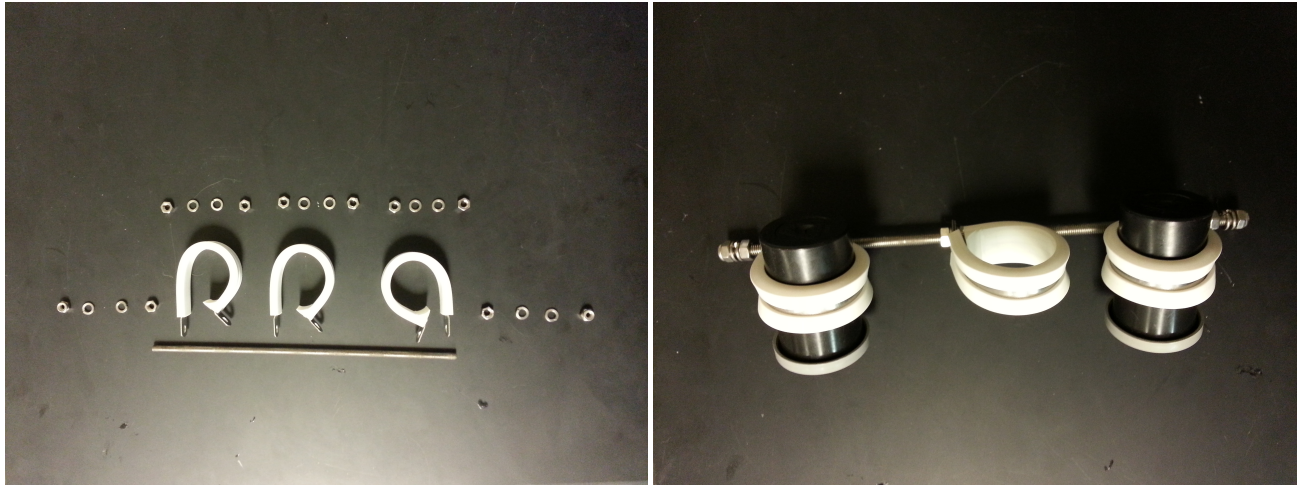


Figure 77: Mounting of the clamps on the rover

Those clamps are also used for adding the extra electronic tube in the middle (the clamp in the middle in the picture above).

To allow more weight to be added to the ROV, its buoyancy needs to be elevated. To do this, watertight tube can be added to the top of the ROV. They can't be placed like the weight containers at the bottom of the roV as it would turn upside-down. One tube as been added at the back on this setup. Extra buoyancy was not needed but the rover was too back heavy and this extra tube allowed to correct this problem. The extra buoyancy added can be simply compensated by more weight. The additional tube at the back can be seen in the picture below :

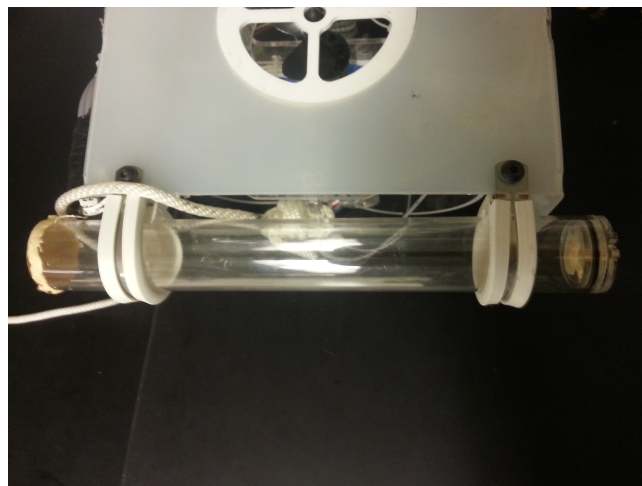


Figure 78: Tube at the back of the roV

## 11.4 External electronic container

Electronic devices that are not waterproof might be added to the rover. In this case, extra containers need to be designed. As the OpenROV is designed to go up to 100m deep, those containers need to be able to withstand a corresponding pressure of 11 bar.

### 11.4.1 Material and shape

Different materials need to be evaluated to build a waterproof case. The capability to withstand pressure can be easily calculated for cylindrical containers as they are often used for pressure vessels. This makes it a good choice of shape for this container.

2 stresses affect the cylindrical pressure vessel : the longitudinal and the hoop stresses. However, the hoop stress being twice that of the longitude stress, the hoop stress only determines whether or not the vessel is capable of handling the water pressure.

Different formula are used whether it is a thin or a thick wall vessel. A vessel is considered thin walled if its radius is larger than 5 times its wall thickness, which is the case here.

The formula for hoop stress (which is valid for both internal or external pressure) is:

$$\sigma_{\theta} = \frac{p \cdot D}{2t}$$

$\sigma_{\theta}$  = hoop stress [Pa]  
 $p$  = pressure [Pa]  
 $D$  = Mean diameter [m]  
 $t$  = Wall thickness [m]

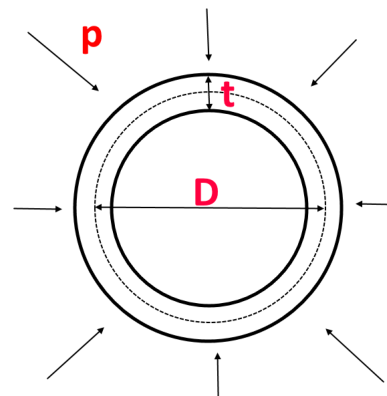


Figure 79: Hoop stress for a cylinder

The external pressure on the cylinder is determined by the water column that can be calculated by the following formula :

$$p = h \cdot \rho \cdot g$$

$p$  = pressure (Pa)  
 $h$  = height of fluid column  
 $\rho$  = density of liquid [kg/m<sup>3</sup>]  
 $g$  = gravitational constant [m/s<sup>2</sup>] = 9.81

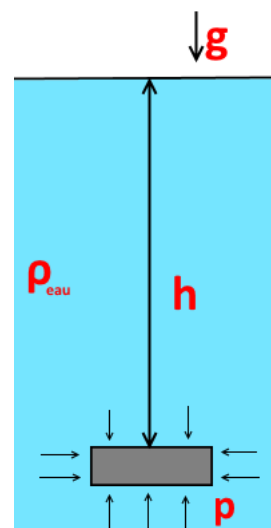


Figure 80: Hydrostatic pressure in the water column

The density the water is variable, depending of the temperature and the amount of dissolved solid but can be without problems approximated by 1000 [kg/m<sup>3</sup>].

the maximum high for the water collumn is for our case 100m (OpenROV design).

the maximum pressure on the tube is then :

$$p = 100 * 1000 * 9.81 = 981000 [Pa] \quad (8)$$

The hoop stress can be calculated with this pressure and the dimensions of the tube chosen. The tube chosen has an inside diameter of 28.6 mm and an outside diameter of 31.8 mm which gives an hoop stress of

$$\sigma_{\theta} = \frac{p \cdot D}{2t} = \frac{981000 \cdot 30.2}{2 \cdot 3.2} = 4.63 \text{ Mpa}$$

Once the hoop stress is calculated it can be compared with the tensile strength of the desired materials.

Two material where compared: PETG and Acrylic. This choice of material was done because there are used for other part of the ROV. This allowed to test if those material where effectively a good choice for the rover and continue the design 'on the same ligne'.

In the worst cases, the tensile strength is 26 Mpa for the PETG and 19.3 Mpa for the acrylic. Although both materials can handle the hoop stress of 4.63 Mpa, the PETG tube was chosen. The reason being that the acrylic is more fragile than the PETG. However, endcaps will be built out of acrylic since it is simpler to handle than PETG.

### 11.4.2 Mechanical design

Once the material used was chosen, the tube containing the electronic needed to be closed hermetically, with wires going in the inside. To do so, a design similar to what was used for the OpenROV main electronic tube can be done. Wires can pass through a hole in the endcap, which can then be hermetically sealed with underwater epoxy.

Different alternatives for the design were made. The first idea was to cut all necessary parts in acrylic sheets, as it is done for the OpenROV. The elements of the resulting design is shown in annexes L

All designed pieces can be cut out of a 3mm acrylic sheet and glued together with acrylic cement, to the design shown in annexe M and in the figure 81.

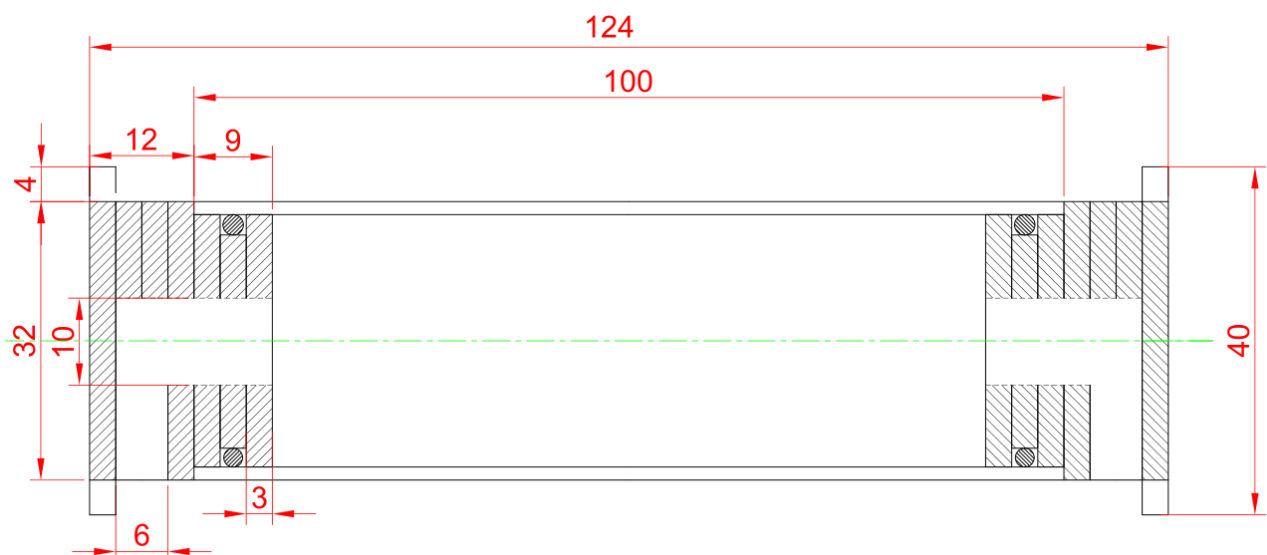


Figure 81: Assembly for the waterproof container

The wires can go through the holes visible on the picture above, out in the water. For connector too big for the holes, like it was the case for the BNC connector of the Conductivity probe, the endcap can be split in two and assemble around the wires as it will be shown later in this chapter.



This design was however made slightly differently using parts of the OpenROV to do something similar, since no machines were available to build those parts. The parts used are shown in the figure 82 below with their thickness indicated, and are all taken from the OpenROV kit.

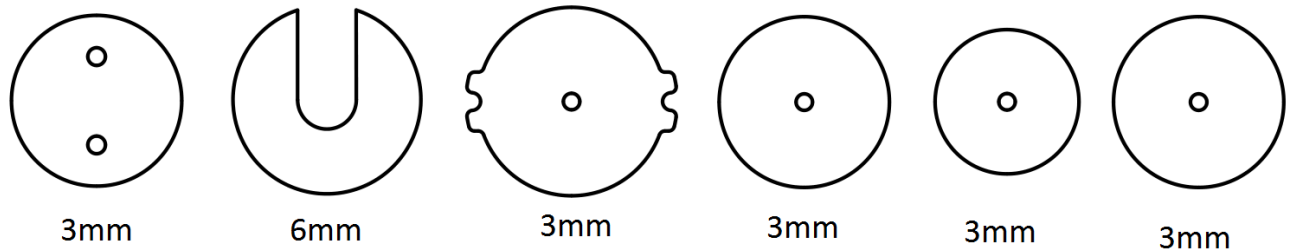


Figure 82: Parts used for the waterproof container

An alternative design (available in annexe N) was also made using a single epoxy bloc, but no machines were available and this design wasn't used.

#### 11.4.3 Endcaps assembly

By lack of material, the endcaps were assembled using spare plastic pieces as shown in the picture 82.

To build those endcaps, a problem was to allow The BNC connector to pass without remaking it. To do so, the parts were assembled in two sets, as shown in the figure 83.

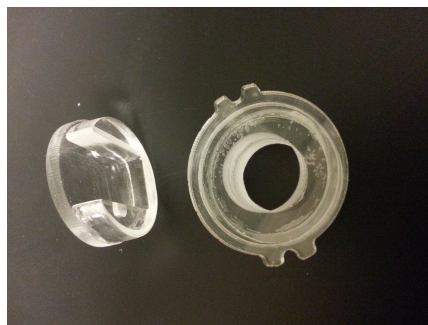


Figure 83: Endcap in two parts

Once it's done, the cable can be placed between the two parts of the assembly as shown in the figure 84, and the two parts can be glued together to build finish the assembly in the figure 85.



Figure 84: Endcap in two parts with cable

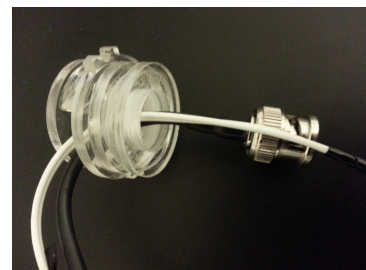


Figure 85: Assembled endcap

The same process has been done for the second endcap, but the wires can be added after the assembly is assembled since no connector are attached.

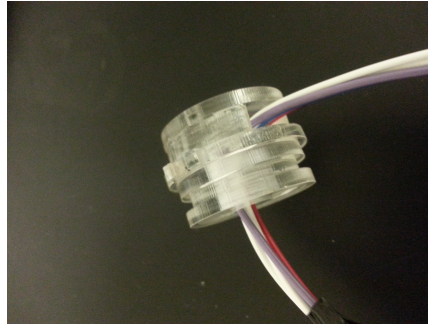


Figure 86: Assembled endcap 2

Once the endcaps are assembled, the wires way-through can be filled with underwater epoxy to seal them watertight. However, the same epoxy than the one used for the main electronic tube's endcaps (see section 11.2) caused troubles. The epoxy couldn't bound to the coaxial cable, which lead to leaks. An other epoxy had to be used : McMaster-Carr Underwater epoxy syringe 7456A53

This epoxy is thicker but bounded to the coaxial cable which allowed to use it for the seal.

#### 11.4.4 Fixation of the tube

The tube is attached to the ROV the exact same way than the weight containers as shown in figure 77. The tube is fixed in the middle of each of both threaded rod.



#### 11.4.5 Final setup

The final setup of the rover can be seen from a front and a side view in the figure 87 below, with the position of each probe labelled.

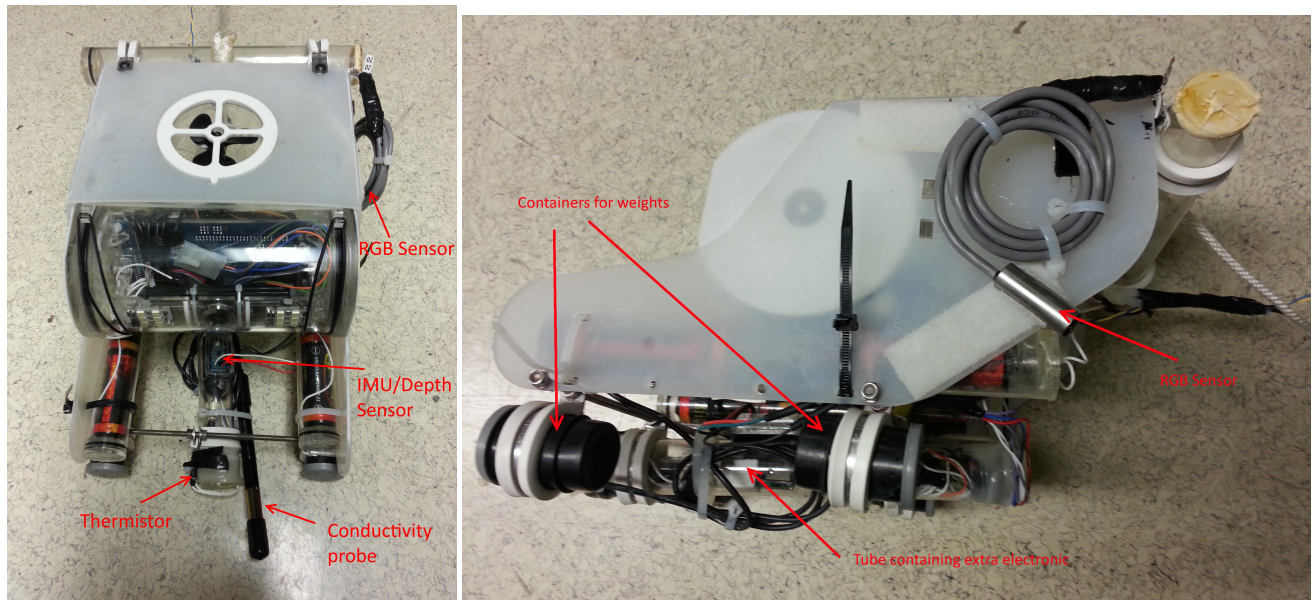


Figure 87: Final setup of the rover

The additional electronic for both the temperature and the conductivity measurement are placed into the small tube in the center of the rover, between the 2 battery tubes and can be seen labelled in figure 88

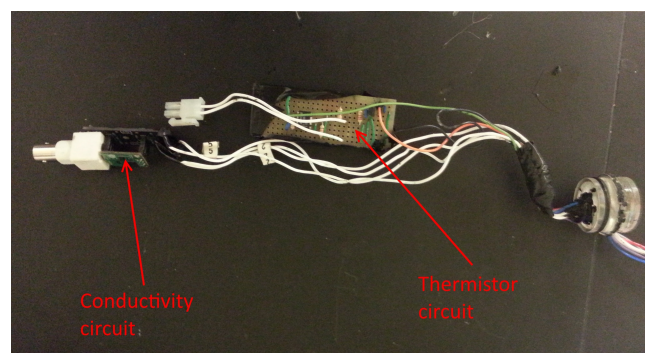


Figure 88: Electronic inside the central tube

## 12 DATA ANALYSIS PROGRAM

### 12.1 Introduction

Data gathered by the rover are stored into a database that can be downloaded for further analysis, but the database format is not adapted to be simply analysed by any user. It is then necessary to develop a program in order to make this analysis easier. This program needs to have a structure that allows to interpret data from new sensors with a minimum of modifications, which is greatly helped by the use of a database that forces a defined common structure for every sensors. The structure of this program should also allow the addition of specialized analysis for further development, since each sensor could provide informations that needs specific calculations or graph formatting to be understood.

The need of various views leads to an interface of type MDI (Multiple documents interface). This type of interface allows to easily handle each new part of the program independently as a new window. On the other hand this type of interface greatly increase the complexity of the general structure: Communication between each window can become complicated due to circular inclusions. It is then necessary to develop a structure facilitating this kind of communication.

It was decided to develop this program in C++, using QT. Qt possesses a large amount of classes that makes the development of graphical interfaces easier and is available for free with few limitations (It is licenced under GNU General Public License version 3 and GNU Lesser General Public License version 2.1).

One of the most important feature for this program is to be able to draw graphs. This was made using the library QCustomPlot for QT which allows to draw a large range of different type of graphs and dispose of a multitude of options for the personalization of the plots.

### 12.2 Architecture

The architecture of the program is composed of two distinct parts.

The first part includes the main elements of the interface, which are basically the elements that are called by the file `main.cpp` and are displayed when the program is opened. Those classes are the following :

- ◆ The class `MainWindow` that forms the main displayed window and the main menu. It also manages some of the actions performed from the menu such as exportations and the opening of the database.
- ◆ The classes `Calendar` and `Measuresets` are docks that allows to view and select measures available in the currently opened database. The class `Measuresets` displays all available measures and `Calendar` allows to filter the results to specific dates.
- ◆ The class `Measures` corresponds to the MDI area and manages all the sub-windows.

The communication between those classes must be bi-directional, which can be tricky in some situation, causing circular inclusion. To make this communication possible and easier to manage, a pattern called `Mediator` was used. The Class called mediator is called through its interface when an action must be performed, and forwards the messages to all subscribed classes. It is very similar to the Subject/Observer pattern but all messages are sent to the mediator and all direct messages between classes should be avoided. Moreover, the mediator can also contain logic that control the program as a whole. In the `mediator pattern`, each class is derived from a class called `Colleague` in which function that can be called from the mediator are defined. Each class derived from `Colleague` must have a reference to the mediator interface `MediatorI`, in order to send messages to the other classes.

More details about the implementation of this pattern are given in the section 12.3.

the class diagram corresponding to the main elements and the mediator pattern is shown in the figure 89:

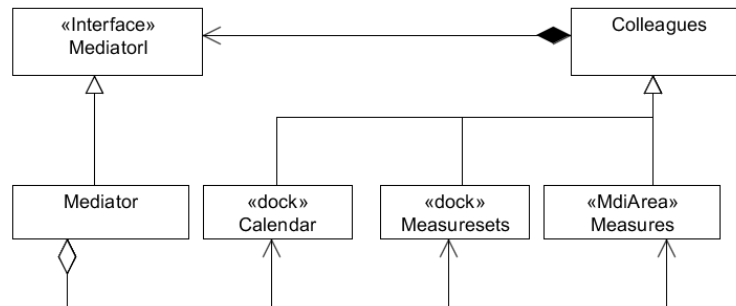


Figure 89: Class diagram for the main elements of the program and the mediator pattern

Once this general architecture is defined, it becomes easy to add new 'modules' to the program. The general implementation of a module can be done in different ways, but must always be created by the `Measures` class which handle their creation and destruction. Each developed module will be explained separately in specific sections of this report.

the class diagram can then be extended for an added module as shown in the picture below :

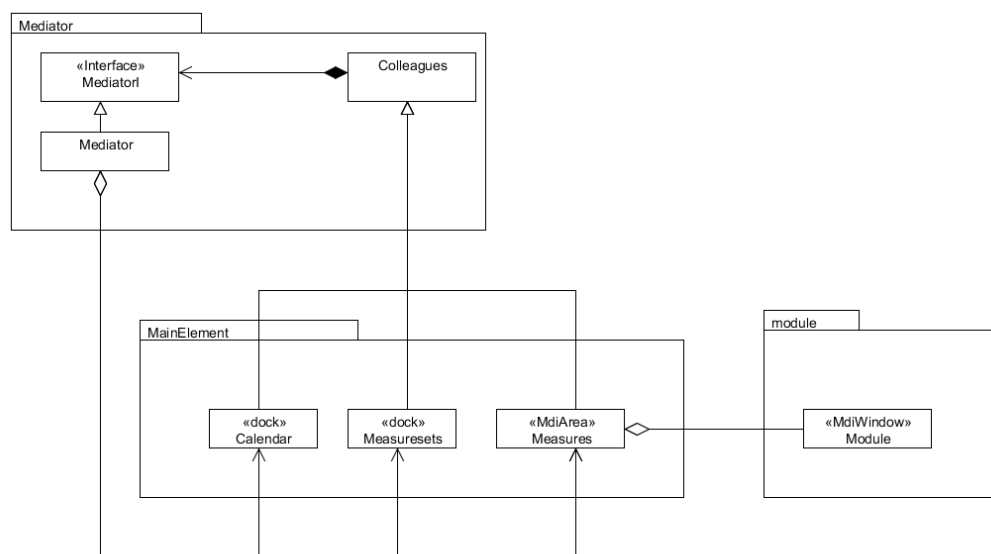


Figure 90: Class diagram with one added module

Each module could be subscribed to the mediator pattern through the `Measures` class in order to receive or send message to other class if necessary . However, this wasn't done for the ones that were implemented to simplify the program, because it wasn't necessary and doing so for too much classes can become difficult to manage. However, if doing so, messages between the module classes and the `Measures` class should be avoided since it breaks the mediator pattern rule that no direct messages should be sent between the colleague classes.

The last aspect to plan for this program is the database connection and management. It has been decided to create one object for it from `main.cpp` and then pass its reference through all classes that needs it. More details will be explained in a section 12.5.

In general in this architecture, it can be seen that the logic is not separated from the view. The reason being that the logic is minimal in this program, that handles almost only the display of data. Designing the program for a full separation would greatly increase the complexity with few advantages. A small separation has however

be considered by implementing the mediator pattern, that could be improved as a kind of controller in further developments.

Only two modules were developed during this project :

- ◆ The package `Measure` that allow to visualise all parameters taken during a measure (dataset) and display graphs showing how each parameter evolved during the time of the measurement as well as statistics about them.
- ◆ The package `Measurescomp` that allows to select two parameters that are compared together in a scatter plot.

The full class diagram containing all implemented modules can be seen in the annexe H

## 12.3 Mediator pattern implementation

Implementing the mediator pattern in QT needed some additional thought. QT uses a system of signal/slot that is very handy to write a simple and clean code for the communication between classes. However, the implementation of this pattern with this system needs a small adaptation in order to work properly.

The signal/slot system require to connect a signal to a slot by giving references to the sender object, its corresponding signal, the receiver object and its corresponding slot. As in the code shown below :

```
1 connect(&sender, SIGNAL(sender_signal()), &receiver, SLOT(receiver_slot()));
```

The mediator pattern requires that a colleague uses an interface to access the mediator, calling the concrete mediator functions by polymorphism. Directly giving the reference to the mediator interface doesn't work with the signal/slot system, because the compiler doesn't see that the mediator is also of type `QObject` (which is necessary to use the signal/slot system). The solution is to cast the mediator pointer into an `Object` pointer, which allows the compilation to be done successfully.

The connection to call a mediator's function can be done as in the following code :

```
1 connect( &sender, SIGNAL(sender_signal), dynamic_cast<QObject*>(mediator) , SLOT(mediator_slot()) );
```

## 12.4 Interface

The main interface of the program is shown in the figure 91 below. Multiple Documents can be opened, displaying different informations available in the opened database file.

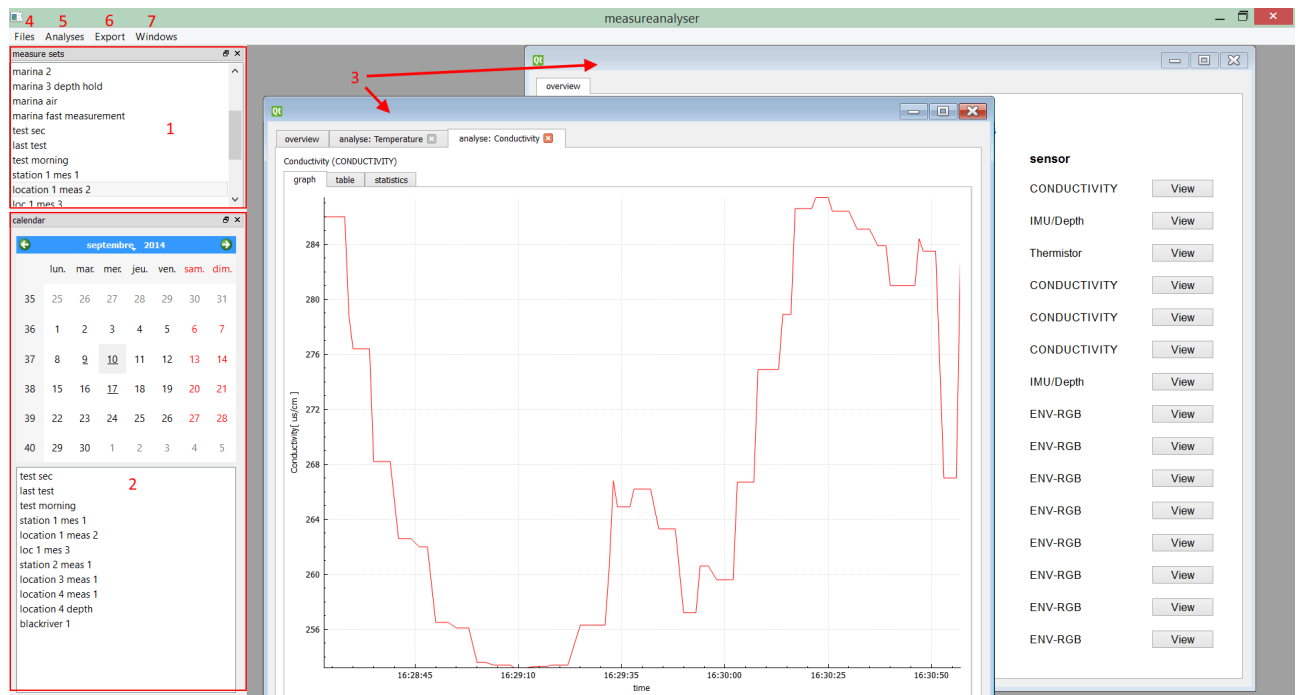


Figure 91: Main interface for the data analysis program

The side bar on the left is composed of two small modules allowing to visualize the different measurements taken by the rover. In the module 1 on the figure 91, all the measures taken are visible. The module 2 allows to display only the measures taken at a specific date, which can be handy if measures are taken over a large period of time.

Double clicking on one of the measures in the side bar will open its corresponding windows, as shown at the number 3. The window will open at the tab overview from which all the measured parameter can be visualized in separated tabs. Multiple windows corresponding to different measures can be opened. All Windows opened this way are `Measure` Objects that are part of the module `Measure`. More information about the interface for each module will be given in their corresponding sections in this report.

On the upper part of the main window, in the Menu bar, various actions can be performed. The menu `files` (4) allow to open and close the database files, the menu `Analyses` (5) allows to open new windows corresponding of specific analysis, in the menu `export` (6), data can be exported to specific formats and in the the menu `windows`, it is possible to interact with all the opened windows to facilitate their management.

## 12.5 Database

### 12.5.1 Creation and utilisation

The class `Sqliteconnector` handling the database is created in `main.cpp`. The `Sqliteconnector` class uses the class `QSqlDatabase` to establish the connection with the database file and the class `QSqlQuery` to execute Sql queries on it.

When starting the program, the user must open a database file. This can be done by using the menu `files->Open`, which calls the method `Openfile()`. In this method, a window allowing the user to choose a `.db` file in his filesystem is displayed. Once a file is chosen, the method `databaseconnection(QString databasepath)` is called, which attempts to open the file. If this method succeed in opening the database file, a global message will be sent through the `Mediator` for all classes and measures availables in the opened file will be visible in the dock part of the program.



### 12.5.2 Data access methods

Most of the methods implemented in the `Sqliteconnector` class are accesses to specific informations in the database. The methods were added one by one when specific data were needed for the program. The return format varies depending on the request but are generally `QMaps` for specific entry and `QVector` containing `QMaps` when multiple entries are returned.

### 12.5.3 Future development of the class

The method `IsConnected` of this class has not been completed. It has been created in order to avoid unexpected errors if a disconnection happened during the use of the program, but only return true in the current version. This might cause some unexpected behaviours in the program, what wasn't however noticed during its use.

## 12.6 Measure module

### 12.6.1 Architecture

The first module that was created represent a measure taken by the rover. It is the window opened when a specific measurement is double-clicked on in the dock area.

This module is composed of various class that form the package `Measure` :

- ◆ The class `Measure` is the controller and the window. It is essentially charged of the management of the tabs
- ◆ The class `Overview` is a tab that provides general information about the measurement (location, operator, time, sensors and parameters used). The user can also select specific parameters to analyse from this tab
- ◆ The class `Analyse` is a tab that provides general information about a specific parameter. It contains different tabs that displays various informations: a graph of a parameter's evolution in function of the time, a table of the values and a statistical plot.

The Plots visible in the `Analyse` tab are not part of the `Measure` package but are located in the `Plots` package. The reason for this choice is that similar plots could be used by other packages.

The measure package is represented in the UML class diagram below :

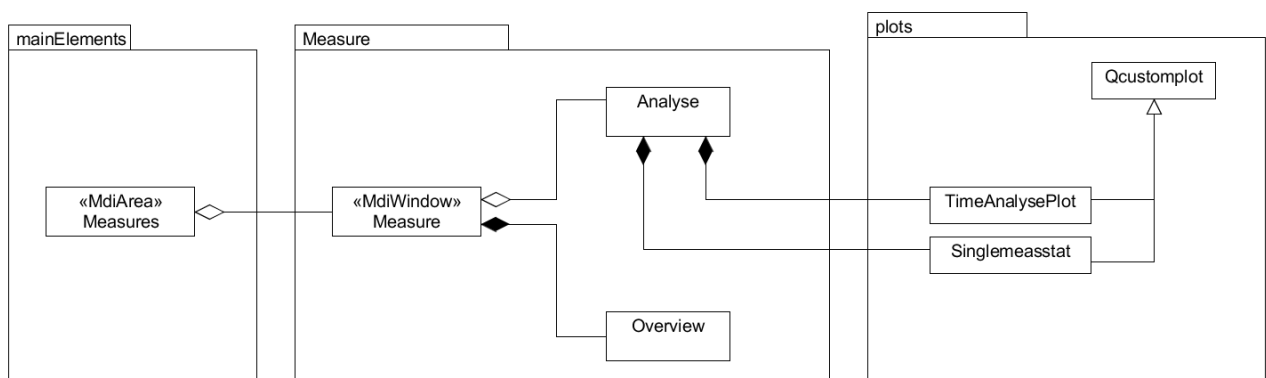


Figure 92: Class diagram for the measure package

The `Measure` class is only managing the tabs. The `Overview` tab is created in its constructor and everything will be managed from it. the `Measure` class opens a new tab when a signal is received through a specific slot.

For example, the slot `newanalyse(int, int)` is called when a parameter needs to be visualise and will then create an `Analyse` Object and add it as a tab.

### 12.6.2 Overview class

This class corresponds to the first tab that is opened when a measure is visualized. It appears as in the figure 93.

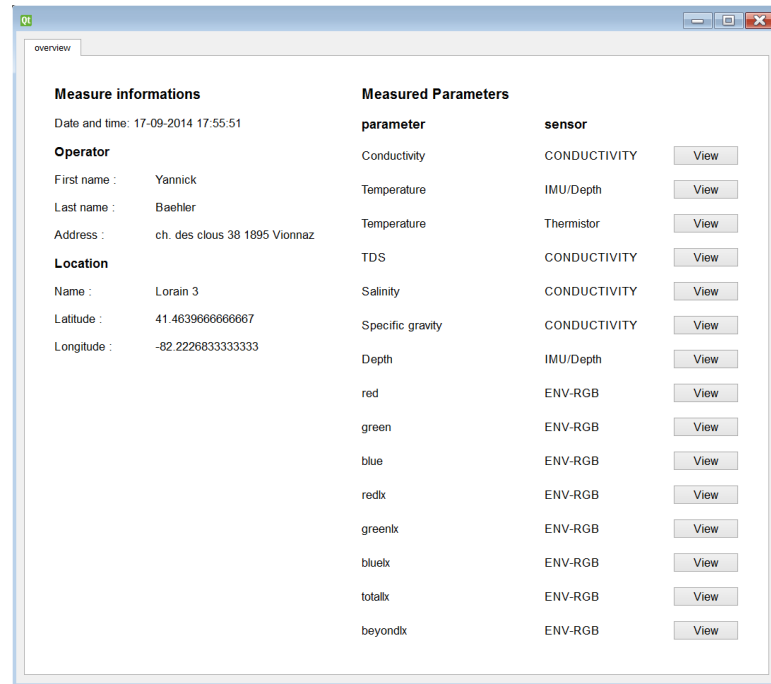


Figure 93: Window opened when a measure is analysed

On this window, the information concerning the selected measurements is shown on the left part. On the right part, the parameters available for this measure are displayed and corresponding buttons allow open an analyse for them.

All displayed elements in the corresponding tab are created and displayed in the constructor of this class. In order facilitate the understanding of this class, a few private methods were written and are called from the constructor to realize the necessary tasks.

- ◆ The method `void initlayout()`; handles the organisation of the tab's layout
- ◆ The method `void initinformations()`; handles the elements corresponding to the left side of the tab (informations about the measurement).
- ◆ The method `void initparametersensors()`; handles the elements corresponding to the right side of the tab (informations about the parameters).

The QT signal/slot system is used in this class in order to forward the click on one of the parameter's button to the `Measure` class that is charged to create the corresponding `Analyse` tab. This can't be done simply because of the dynamic aspect of the buttons creation (the number of parameters vary depending on the database file). However, Qt provides classes that help doing this by allowing to link multiple signals to one slot.

In the `void initparametersensors()` method, each created button is linked to a vector containing its corresponding parameter and sensor by using a `QSignalMapper` object (`btnmap`). Each button signal is then connected to this `QSignalMapper`'s `map()` slot.

```

1  param = new QVector<int>;
2
3  param->append(params.at (var) .find("parameterid") .value() .toInt()); //get id param
4  param->append(params.at (var) .find("sensorid") .value() .toInt()); //get id sensor
5  btnmap.setMapping(btn, (QObject *) param);
6  connect (btn, SIGNAL(clicked()), &btnmap, SLOT(map()));

```

Once all button are created, the QSignalMapper mapped(QObject \*) signal is connected to the newanalyse() slot that is charged to alert the Measure object about which parameter needs to be analysed.

```

1  connect ( &btnmap, SIGNAL(mapped(QObject *)), this, SLOT(newanalyse(QObject *)) );

```

When the newanalyse(QObject \*) method is called, The QObject \* parameter needs to be re-casted into a QVector<int> \*, which contains the information about the button that was clicked on. The corresponding values can then be sent to the Measure object by emitting a signal.

### 12.6.3 Analyse

An Analyse object is linked to a specific parameter and a specific sensor. An object of this class is created when the Measure object newanalyse(int parameter, int sensor) slot is called. Once created, the object is added as a tab and is displayed as in the figure 94

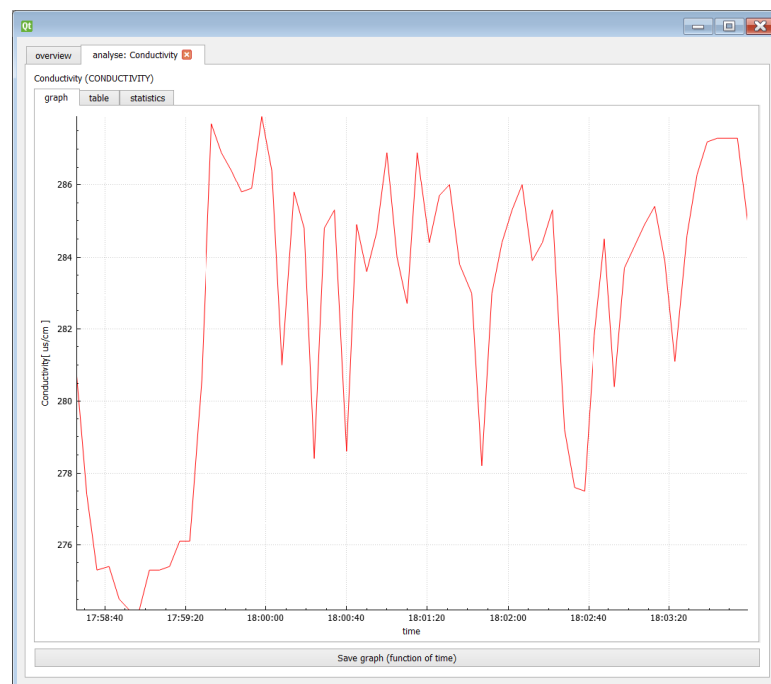


Figure 94: Tab opened when a new analyse is launched

This tab is composed of three sub-tabs: One displaying the graph of the evolution of the parameter over time (figure 94), an other allowing to visualize the values in a table, and a last one displaying a statistical plot of the parameter's values during the measurement.

In the constructor of this class, multiple method are called in order to organize the code (as done for the Overview class).

- ◆ The method `initlayout()` ; handles the organisation of the layout.
- ◆ The method `initvalues()` ; performs database requests in order to get all the parameter's values and store them into Vectors and Lists for further processing



- ◆ The method `initgraph()` ; handle the creation of the `TimeAnalysePlot` that plot an analyse of the parameters over time.
- ◆ The method `inittable()` ; handle the creation of the tab containing all values displayed into a table.
- ◆ The method `initstatplot()` ; handle the creation of the tab containing the statistical plot.
- ◆ The method `layoutsetup()` ; places the different elements in the layout created with `initlayout()` ;.

It must be noted that the plots themselves are not part of this class but are defined in the `Plot` package of the application.

A button placed at the bottom of the tab allows to save the analyse of time's plot. Clicking on this button will call the slot `savegraphtofile()` that handles this functionality. The positioning of this button is not optimal since it is visible from all tabs, but has only a use for one. The layout should be modified to correct this.

## 12.7 Paramaters comparison

### 12.7.1 Architecture

The second module that was created represent a comparison of two parameters taken during a measurement. The current design of this module is very simple. It was initially thought to be more in depth but it couldn't be finished during this project and only very basic functionalities were implemented in a quick manner.

This module is composed of various class that form the package `Measurecomp` :

- ◆ The class `Measurecomp` is the controller and the window. It is essentially charged of the management of the tabs
- ◆ The class `Viewsettings` is a tab that allows to select the parameters that need to be compared
- ◆ The class `Analysecomp` is a tab that contain the analyse of the comparison selected in the tab `viewsettings`

The Plots visible in the `Analysecomp` tab are not part of the `Measurecomp` package but are located in the `Plots` package, in the exact same way than the plots in the `Measure` package.

The `measurecomp` package is represented in the UML class diagram below :

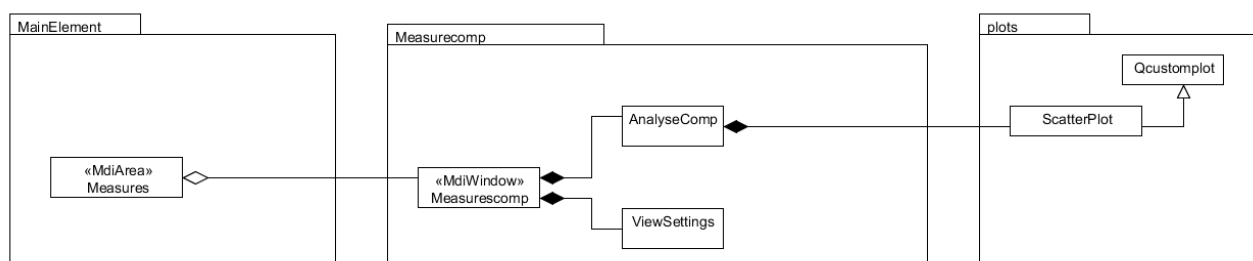


Figure 95: Class diagram for the `measurecomp` package

### 12.7.2 Viewsettings class

In the constructor of this class, a form is created that allows to select 2 parameters to compare, which will be plotted in a graph. This form is shown in the figure 96 below :

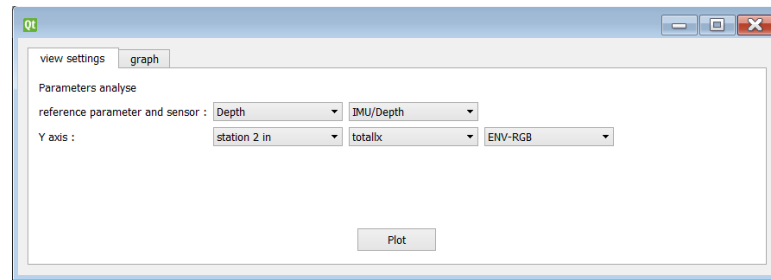


Figure 96: Tab opened for the choice of the parameters to compare

The lists allowing to choose the different elements needs to be updated depending of the element selected in the list placed before (when a parameter is chosen, the list of sensors must be updated to correspond only to those available for this parameter).

To do this, each list is linked to a corresponding function in the class through the QT's signal/slot system. When the value of a list is changed, the other lists are updated according to what was chosen.

### 12.7.3 Analysecomp class

This class is a simple tab, that simply displays the graph corresponding to the two parameters to analyse, chosen from the class `Viewsettings`. The graph plotted can be seen in the figure 97 below.

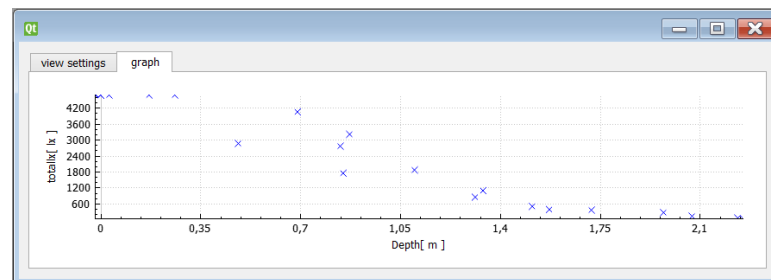


Figure 97: Tab opened with the graph of the sensors comparison

The current version of this class only display the object `Scatterplot` and forwards data to it if some are received for an update.

## 12.8 Plots

In order to make plots available for other class than the one there were originally designed for, they are all designed in specific classes and are located into the package `Plots`.

All plots are drawn using the library `QCustomPlot`, also located in the same package.

The plots classes implemented are :

- ◆ `TimeAnalyseplot` that is used by the package `Measure` in order to plot the evolution of a parameter during a measurement.
- ◆ `Singlemeasstat` that is used by the package `Measure` in order to plot a statistical plot of a parameter during a measurement.
- ◆ `Scatterplot` that is used by the package `Measurecomp` in order to plot a parameter in function of an other in a scatter plot.

Those classes are all derived from the `QCustomPlot` object.

The implementation of those classes is very simple in their current state, since not a lot of modifications were done comparing to the standard `QCustomPlot` plot. They simply implement a configuration characteristic of their corresponding graph in their constructor. `Singlemeasstat` and `Scatterplot` also have a functions implemented allowing to renew the data to plot.

## 12.9 Google Earth exportation

Google earth was often used to visualize the path taken by the plane when taking picture and matching the interesting images to their location. Using Google earth allow to have a good visualisation on a map without developing specific software. For those reasons, It has been chosen to develop an exportation under a format that can be opened by Google earth. A good format to do so is the `KML` file, this format is base on the `XML` format but is specialized for Google earth.

The main structure of a `KML` file is the following :

```

1 <?xml version='1.0' encoding='UTF-8'?>
2 <kml xmlns='http://www.opengis.net/kml/2.2'>
3   <Document>
4
5   </Document>
6 </kml>

```

Each location on the map can be declared between the `Document` tags. A lot of different tags can be used for each location allowing a large range of options. The minimum required for placing a location is the following structure :

```

1 <Placemark>
2   <name></name>
3
4   <description>
5
6   </description>
7
8   <Point>
9     <coordinates>xx.xxxxx,xx.xxxxx</coordinates>
10  </Point>
11 </Placemark>

```

Each `Placemark` corresponds at a measure that was taken by the rover, which always has a name. The measures that are interesting to display on Google Earth have also always coordinates that must be entered in a decimal format in the order longitude, latitude between the tags `coordinates`.

All informations on the measurement can be added between the `description` tags. The `html/css` is supported which allows to display the informations as wanted.

The exportation is launched from the menu `Export`, that calls the function `togooglemap()` in the class `MainWindow`, which handles the exportation. It uses a `QFile` object to create the file and a `QTextStream` object is used to parse all informations to be exported.

## 12.10 Conclusion

The Analysis program fills its role to analyse the data collected by the rover. The database can be imported, all data can be visualised and plots are drawn successfully. The structure implemented for adding module easily could be tested by implementing two modules in the current version, and works well.

However, some necessary corrections should be done on a few parts:

- ◆ The part concerning the scatter plots (see section 12.7) was developed only shallowly and could be greatly improved.
- ◆ The database connection functionalities should be checked concerning the risk of disconnection of the database (see section 12.5)

Moreover, no test were done on the software, except simple observation of its proper working. Some more precise tests should be done to ensure that no unexpected behaviour can append. Problems were however not observed during the project.

## 13 FIELD TESTS

### 13.1 Introduction

Several Tests have been made on the system. The Tests specific for each sensors or modification made are written in their specific section of this report. The tests shown in this section concern only the test made with the entire system, on the field.

The Tests made had 2 purposes:

The first one is to test the system in itself and check if software or hardware issues that were not discovered in the laboratory could show up in a full scale use.

The second is to analyse if data gathered by the rover's sensors on the field were exploitable for analysis.

### 13.2 Methodology

Firtly, several locations were chosen using a plane, between Cleveland and Sandusky. Photography from the plane allows to see with precision where algea blooms are and helps to determine different locations where interesting data could be taken.

From those aerial pictures, several places were selected for measures: 4 location in Lorain, 2 locations in the Sandusky bay, 4 location between Sandusky and Kelleys island and 2 locations in Avon. Those location can be seen in the picture 98 below :



Figure 98: Map of the different locations where measurement where taken

Between all those measurements, 4 of them were chosen to demonstrate different environments inside the lake, taking aside measurements taken in rivers or bay. All of these points had at least 7m in depth, which was necessary to have interesting data concerning the RGB Sensor. those point are :



A location in Lorain, a few kilometers from the coast (41.4704°N 82.2272°W) the 10th of September 2014. the water was slightly green looking but no visible sunded solids where visible. the picture 99 below has been tacken at this location.

Figure 99: water at the station 4 near Lorain

A location in Lorain, a few kilometers from the coast ( $41.4639^{\circ}\text{N}$   $82.2226^{\circ}\text{W}$ ) the 17th of September 2014. The water was greener looking than the 10th of September but no visible suspended solids were visible. The water can be seen in the picture 100.



Figure 100: water at the station 3 near Lorain



Figure 101: Water between Sandusky and Kelleys island

A location between Kelleys island and Sandusky ( $41.5383^{\circ}\text{N}$   $82.6344^{\circ}\text{W}$ ) the 19th of September 2014. Small algae were visible suspended in the water. Those suspended algae are visible in the picture 101

A location near Kelleys island ( $41.6014^{\circ}\text{N}$   $82.6691^{\circ}\text{W}$ ) the 19th of September 2014, suspended algae were visible like for the previous location, but there was also presence of a lot of foam at the surface, as it can be seen in the picture 102

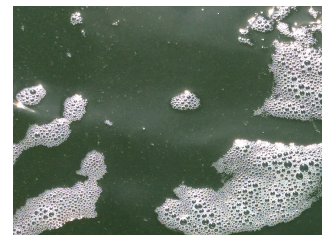


Figure 102: Water near Kelleys island

From those locations, multiple parameters can be compared and will be explained in the sections below:

- ◆ Temperature
- ◆ Conductivity
- ◆ Turbidity
- ◆ Water Colour

All data used in this section were gathered by the rover, but analysed separately from the Analysis program. A useful future development for the analysis program would be to build modules able to plot the graphs showed in this section in order to facilitate this kind of analyse.

## 13.3 Water parameters

### 13.3.1 Temperature

Temperature of the water was taken at the chosen locations in the lake and results can be compared, as shown in the figure 103 below. The different locations had very different results, but it must be taken into consideration that they were not taken the same day, which makes the average value of the temperature irrelevant in this comparison. This can however still be taken into consideration for measurement taken the same day as in figure 104. However, what can be seen here is the stratification of the temperature measurement. Locations

without suspended algae showed very little variations in function of depth, which is not the case in location where suspended algae were visible.

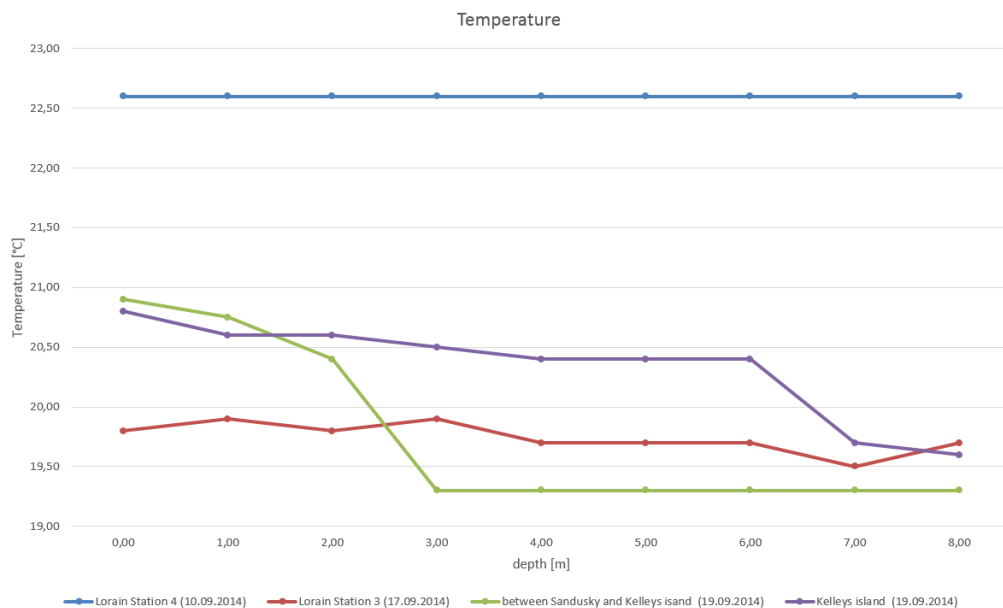


Figure 103: Temperatures at different locations in the lake

It is however important to take in consideration that stratification into lakes is not always due to the presence of algae and can be observed throughout the year. In Summer and Winter the lake temperature tend to be respectively higher and lower at the surface than at the bottom, but comes back to a similar temperature during Spring and Autumn. It is then necessary to take measurement regularly at the same location in order to make a difference between usual seasonal changes and other causes which makes that data taken like in figure 103 at different location must be analysed carefully.

Temperature at the locations near Sandusky seems to indicate a correlation between algae bloom and a higher temperature. Which makes sense since suspended solids into water tends to absorb more heat [8]. This can be supported by measurements taken the same day in the bay, where suspended algae were not there. The temperature there was significantly lower than in the lake, despite of being shallower and contained. The difference in temperature in all location taken the same day near Sandusky can be seen in the picture below :

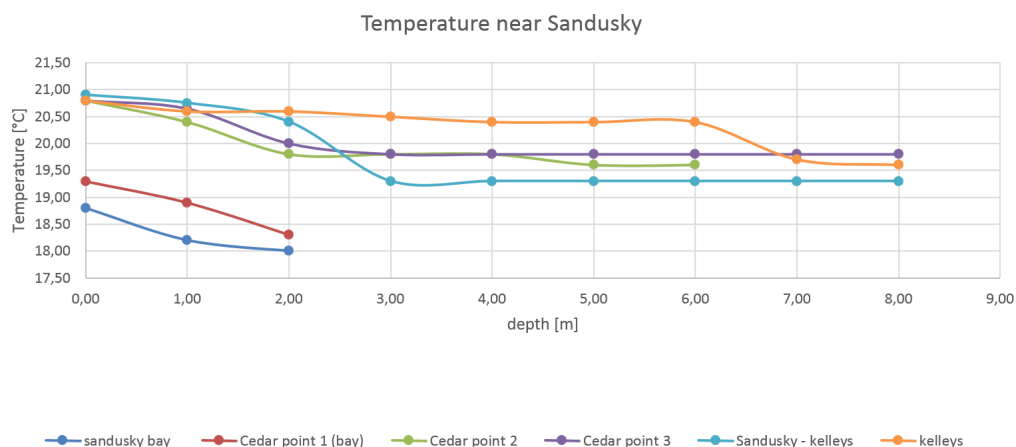


Figure 104: Temperatures in Sandusky bay and between Sandusky and Kelleys island

The difference in the stratification between different locations where algae are present, is difficult to explain and no source where found about this.



### 13.3.2 Conductivity

Conductivity was measured at several locations in function of depth as it can be seen in the figure 105. There weren't meaningful results in the values of the conductivity in the lake, since it varied very little in average (between 10 and 15  $\mu\text{S}/\text{cm}$ ) and at different locations.

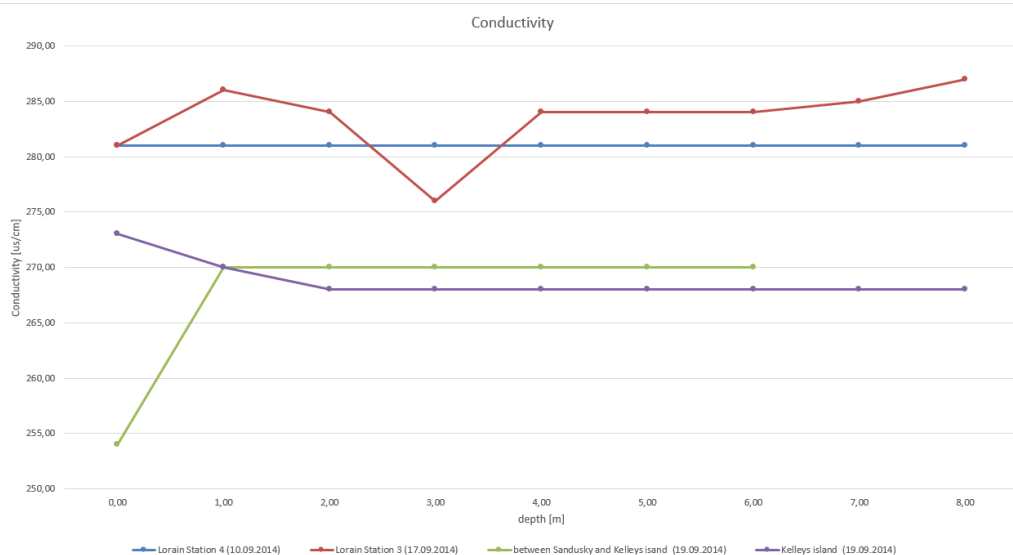


Figure 105: Temperatures at different locations in the lake

However, the value of the conductivity has shown high instability in environment where algae were present, as well as a small stratification. This stratification is however difficult to analyse since the two locations where algae were totally different. The location between Sandusky and Kelleys island had a lower conductivity at the surface and the location at Kelleys island had a higher conductivity at the surface. In order to make this parameter useful for monitoring the algae growth it would be necessary to take periodical measurements throughout the year and observe the tendencies.

Even if measurement of conductivity didn't give interesting results about the direct observation of algae, some interesting data could be gathered. Measurement were taken in Blackriver, which is one of lake Erie affluent, at two different days and with very different results as it can be seen in the picture below:

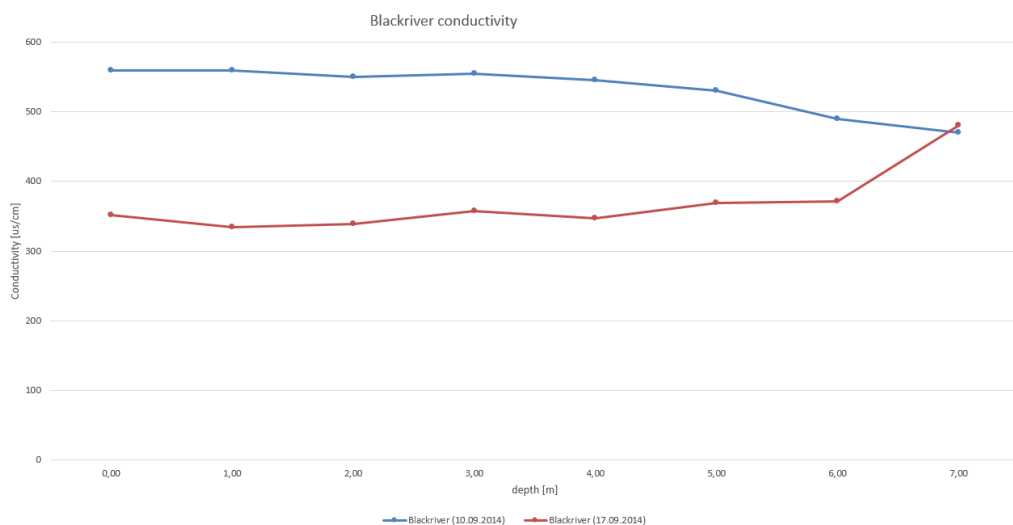


Figure 106: Conductivity in Blackriver



The conductivity in river is known to be highly sensitive to the weather. If it rains, sediments are likely to enter into the river, increasing significantly its conductivity. The measurement taken the 10th of September showed a much higher conductivity as it rained the day before. This was not the case for the 17th of September measurement, which showed a significantly lower conductivity.

### 13.3.3 Turbidity

A measure of turbidity can't be done by the system as a precise measure of the light scattering, but can be correlated to the light attenuation through the water column. In turbid environments, light will decrease faster than in clear water, which allows to make this comparison. The zone called euphotic zone can be determined, which is defined by the zone where the amount of light is above 1% of the surface light.

For each chosen location in the lake, the percentage of remaining light at 6 meters was measured. This gives the following result :

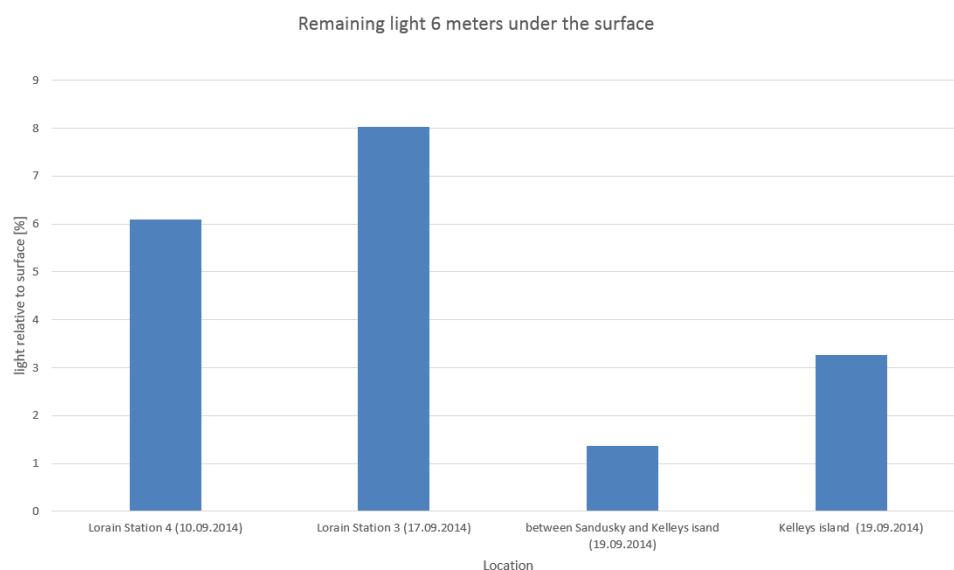


Figure 107: Remaining light 6 meters under the surface at different locations in the lake

It can be seen that, in locations where algae were present, the remaining light at 6 meters is around 2% of the surface lake, for 7% where no algae were visible. Even if this seems to be a very small difference, the light attenuation through the water column is logarithmic which makes this difference of 5% more important than it seems at first glance.

The light in function of depth can be plotted as in the picture 108 below :

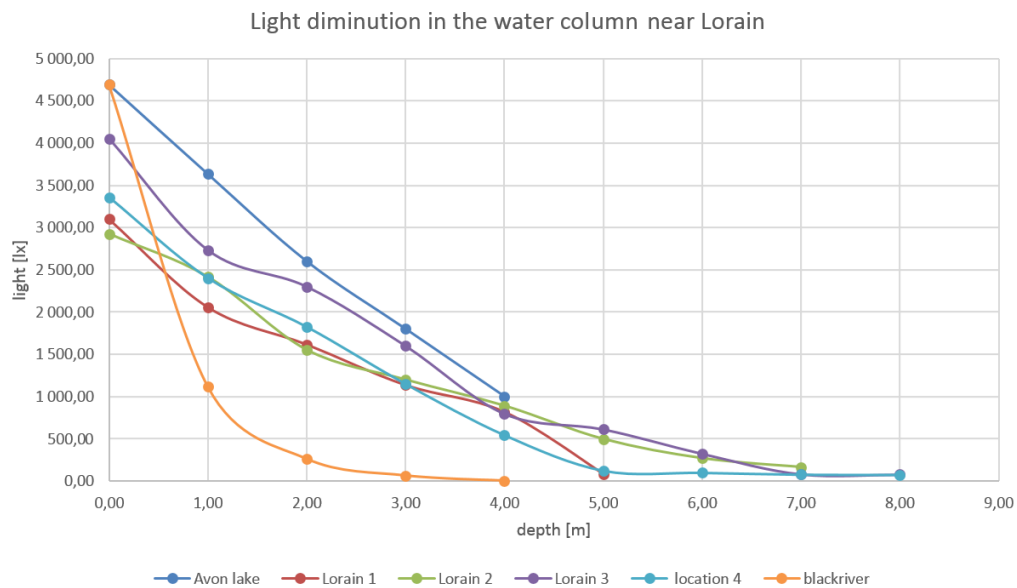


Figure 108: Light in function of depth for locations near Lorain taken the 17th of September 2014

The transparency of water can be evaluated by considering the slope for each locations. Different stations in the lake show a very similar turbidity with their euphotic zone being below 8 meters (the lake wasn't deep enough to measure exactly where the remaining light reached 1%).

The case of Blackriver is totally different, the euphotic zone could be measured to be between 3 and 4 meters below the surface, which shows a much higher turbidity. For comparison with what the water looked like, the pictures below show the rover into both environments :



Figure 109: Water with the rover in Lorain the 10th of September 2014. Figure 110: Water with the rover in Blackriver the 10th of September 2014

In parallel of the light in function of depth, measurements were taken using a Secchi disk. The Secchi depth (depth at which the disk disappears) were recorded at each locations. The euphotic zone can be estimated from those results, by multiplying the Secchi depth by a coefficient. This coefficient is however highly variable but can be considered between 3 and 5 in lakes.

The Secchi depth measurements and their estimated euphotic zone depth (with a coefficient of 4) can be seen in the table below :

Location	Seechi depth [m]	Estimated euphotic zone [m]
Avon lake	2.9	11.5
Lorain 1	3	12
Lorain 2	2.6	10.5
Lorain 3	2.9	11.5
Lorain 4	2.8	11
Blackriver	0.5 ft	2

Those measurement with the disk, show results comparable to the measurement done with the RGB sensor. An exact comparison is difficult, but the result are not at odd with the light measurements and are in the same range.

### 13.3.4 Water Colour

The colour of water has been taken in function of depth, which allows to know which range of wavelength is able to pass through. The different ratio of the red, green and blue spectrum can then be analysed in order to estimate the presence of elements into the water. As explain in section 8.1.2, a good way to detect the presence of chlorophyll is to observe the ratio of the blue light, which should be the less absorbed through the water column except in presence of some substance. It is however important to take in consideration that the chlorophyll is not the only element that would absorb in the blue spectrum.

The figures 111 and 112 show the composition of the received light both near the surface and at 6 meters underwater.

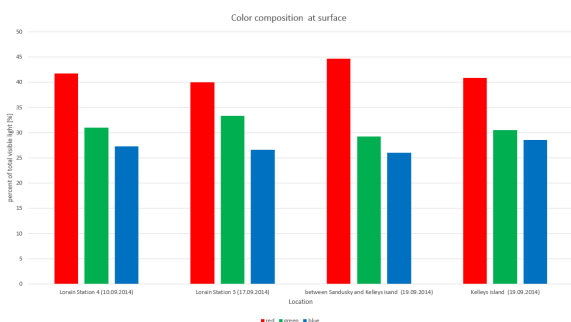


Figure 111: Light composition close to the surface

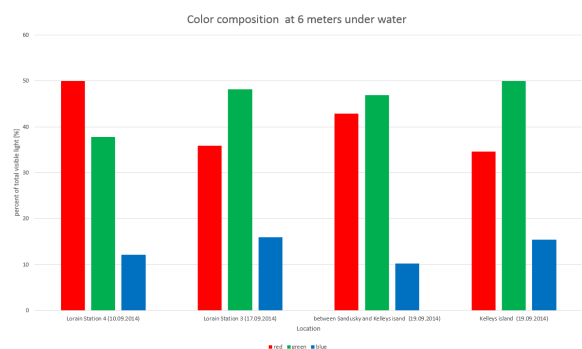


Figure 112: Light composition 6 meters under the surface

It can be seen that in those locations, the absorption of the blue light is highly dominant, and decreases from about 27% of the visible light composition to 13%. This has surprisingly been consistent throughout all measurement and might indicate the presence of chlorophyll. It would however needs further researches to confirm it is really due to the chlorophyll.

## 14 FUTURE DEVELOPMENT

Although the system could be developed to respond to the specifications required and can be used for its purpose, additional development can be done in order to improve the system.

The sensors are all working and their values properly logged, but some additional work and improvements could be done :

- ◆ Improvement of the thermistor circuit.
- ◆ Provide calibration possibilities for the thermistor measurement (web interface, rover software and hardware).
- ◆ Pre-processing for the data gathered by the RGB sensors.
- ◆ Parallelization of the parameters logging in order to avoid the rover to regularly block.
- ◆ Better positioning of the RGB sensor.
- ◆ Implement a possibility to replace easily the database on the rover.

The principal development that could be done in the future is the improvement of the data analysis program. The main structure was implemented and the data gathered by the rover can be visualised properly. However, few test were done to ensure that no unexpected behaviour can append, wich should be done in the future. The parameters comparison module is also only in a embryonic state and needs further development. Some extra modules could also be added to the program in order to improve the user experience, those module could plot the various graph shown in the section 13, that are useful for the analysis of the various parameters.

Unfortunately, the rover had also an accident during the last test. The communication wires were cut and a leak was observed. The wires were replaced, but the connection with the rover often fails since this accident. The problem is explained in the annexe K

## 15 CONCLUSION

The project could be realized to fit the requirements. 3 sensor were added successfully and their data could be gathered and logged into the database.

The conductivity sensor is properly working and calibration can be easily performed by any user through an interface directly implemented into the rover's cockpit web page. The temperature measurement could be taken by a thermistor in order to take quick measurement of this parameter, which could be observed effective and useful. The RGB sensor could be also added and is able to send its data.

Those measurements can also be successfully analysed through the software developed, which allow to visualize them through various graphs.

Various measurement were also taken in the lake in order to verify that all sensor could gather accurate data and give useful information about the water quality through the water column. Those test went well and data can be considerate as exploitable. A few problem are still present and improvements can still be done, but nothing has an impact on the quality of the data taken.

This project could finally show that useful water quality monitoring can be performed through the entire water column for a low cost, and can thus be considered successful for its main purpose to be a proof of concept.

Sion, the 30.11.2014

## 16 REFERENCES

- [1] EPA United states Environmental protection agency, Water: Monitoring & Assessment / Turbidity, <http://water.epa.gov/type/rsl/monitoring/vms55.cfm> (viewed the 20.11.2014)
- [2] EPA United states Environmental protection agency, Water: Monitoring & Assessment / Conductivity, <http://water.epa.gov/type/rsl/monitoring/vms59.cfm> (viewed the 21.09.2014)
- [3] EPA United states Environmental protection agency, Water: Monitoring & Assessment / Temperature, <http://water.epa.gov/type/rsl/monitoring/vms53.cfm> (viewed the 21.09.2014)
- [4] EPA, Status of Nutrients in the Lake Erie Basin, [http://www.epa.gov/lakeerie/erie\\_nutrient\\_2010.pdf](http://www.epa.gov/lakeerie/erie_nutrient_2010.pdf) (viewed the 12.11.2014)
- [5] Kentucky government, Total Suspended Solids and water quality, <http://ky.gov/nrepc/water/ramp/rmtss.htm> (viewed the 21.09.2014)
- [6] USGS, Water properties: Turbidity, <http://water.usgs.gov/edu/turbidity.html> (viewed the 21.09.2014)
- [7] USGS, Water properties: Temperature, <http://water.usgs.gov/edu/temperature.html> (viewed the 21.09.2014)
- [8] Washington State Department of Ecology, A Citizen's Guide to Understanding and Monitoring Lakes and Streams : Lakes, <http://www.ecy.wa.gov/programs/wq/plants/management/joymanual/temperature.html>, (viewed the 24.09.2014)
- [9] Fondriest Environmental, Fundamentals of environmental measurement : Conductivity, Salinity & Total Dissolved Solids, <http://www.fondriest.com/environmental-measurements/parameters/water-quality/conductivity-salinity-tds/> (viewed the 12.11.2014)
- [10] Atlas Scientific, arduino mega EC sample code, [http://www.atlas-scientific.com/\\_files/code/arduino-mega-EC-sample-code.pdf](http://www.atlas-scientific.com/_files/code/arduino-mega-EC-sample-code.pdf) (viewed the 10.08.2014)
- [11] Atlas Scientific, EC EZO Datasheet, [http://www.atlas-scientific.com/\\_files/\\_datasheets/\\_circuit/EC\\_EZO\\_Datasheet.pdf](http://www.atlas-scientific.com/_files/_datasheets/_circuit/EC_EZO_Datasheet.pdf) (viewed the 12.09.2014)
- [12] OpenROV, OpenROV Website, <http://www.openrov.com/> (viewed the 11.11.2014)
- [13] OpenROV, OpenROV Github, <https://github.com/OpenROV> (viewed the 09.09.2014)
- [14] BeagleBone, BeagleBone Black, <http://beagleboard.org/BLACK> (viewed the 10.10.2014)

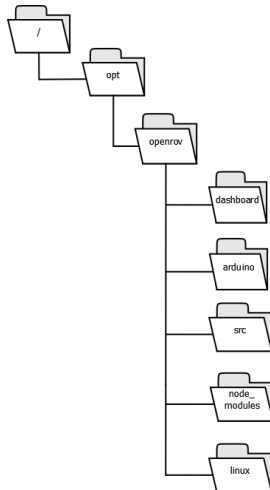


# Appendix

## A DIRECTORIES ORGANISATION

All files that are necessary for the development of the OpenROV are stored in the BeagleBone Black in the directory `/opt/openrov`.

in the folder `openrov`, files are organised in the following manner :



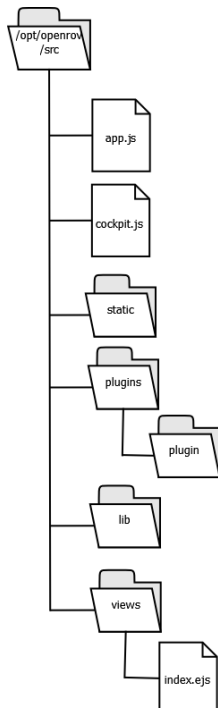
The directory `src` includes all files concerning the cockpit program, which consist of both the `node.js` process and the browser interface for the OpenROV. Further explanation will be given for its file organisation in the next section. The directory `arduino` contains all files that drives the Open-

ROV Controller Board. Those files can be compiled and uploaded on the controller board using the script `/opt/openrov/linux/arduino/firmware-installfromsource.sh`. The directory `dashboard` includes all files concerning

the OpenROV Dashboard which allows to choose between the cockpit interface, the cloud9 environment or activate the samba server. The dashboard is displayed when entering the address of the openROV in the browser (default: 192.168.254.1). The directory `node_modules` includes all the modules used

in the `node.js` program. The directory `linux` contains various scripts for the BeagleBone Black linux system.

Most of the files necessary for further development are present in the `arduino/OpenROV` and `src` folders. The `src` folder is organised in the following manner :



The `app.js` file handles the logs for the cockpit application into the file `/var/log/openrov.log` a

The `plugins` folder includes all additional plugins for the cockpit application. Every plugin must be added as one of its sub-folder and will be added automatically to the cockpit program. The plugins have to follow a specific organisation, which will be explained later in this report.

The `static` folder includes some files for the cockpit interface: The main cockpit program view, the keyboard management, and various other utility files.

The `lib` folder contains the files that handle the communication between the BeagleBone Black and The OpenROV Controller board. It contains also files that handle the camera.

The `views` folder contains an `index.ejs` file that implement the skeleton of the main browser view.





## B OPENROV STRUCTURE DIAGRAM

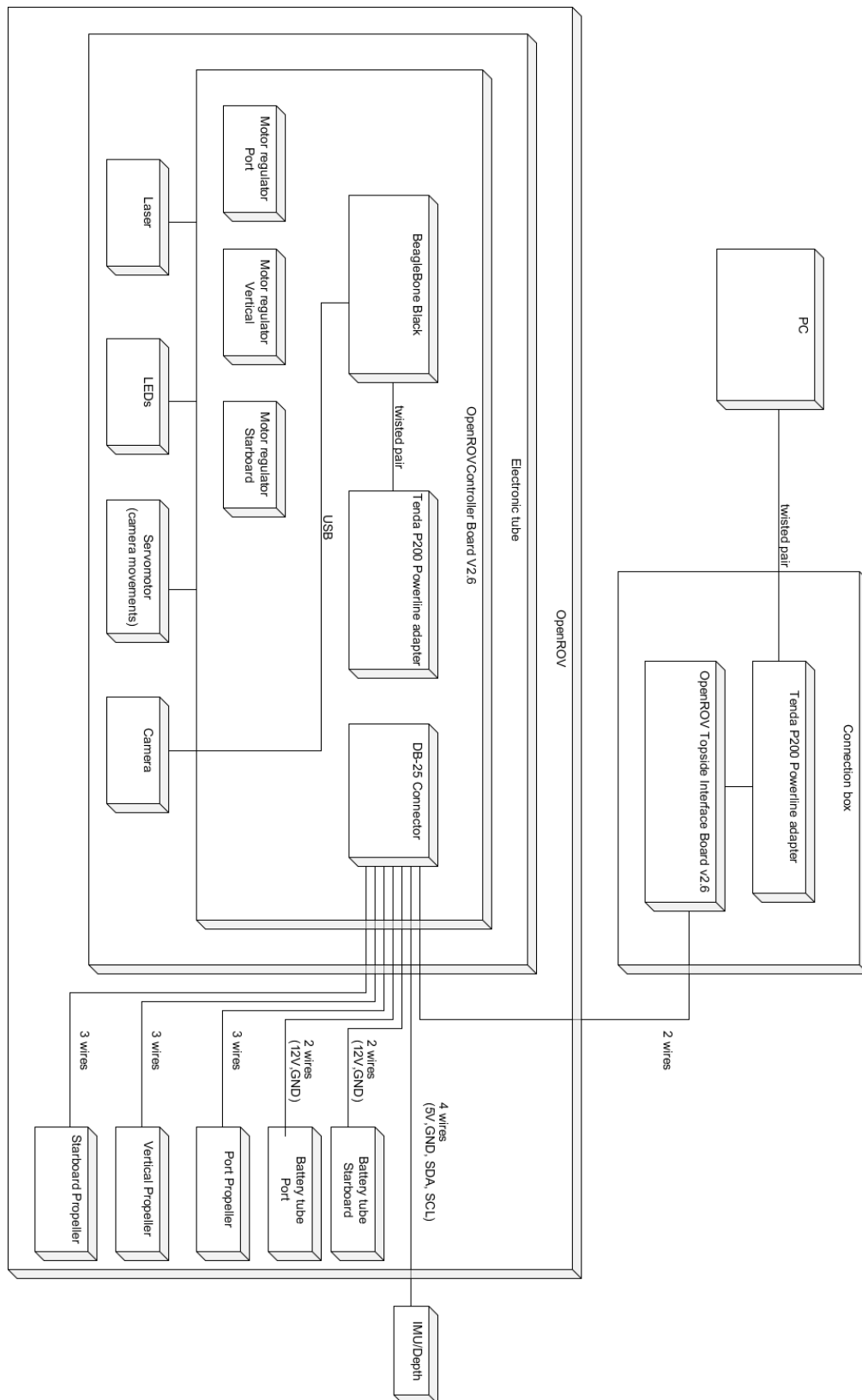


Figure 113: UML physical diagram for the OpenROV



## C CONDUCTIVITY COMMANDS

This table shows the different commands from the conductivity circuit[11] that can be used and how it is translated for the serial communication between the BeagleBone Black and the OpenROV Controller Board.

Conductivity circuit commands	RS232 messages between boards	Description
C,0	condc(0)	Continuous mode disable
C,1	condc(1)	Continuous mode disable
C,?	condc(2)	Query continuous mode
T,?	condtt(1)	Query temperature compensation
T,x.y	condtt(2, x, y)	Set temperature compensation
K,?	condk(0)	Query the probe K constant
K,0.1	condk(1)	Set the probe K constant to 0.1
K,1	condk(2)	Set the probe K constant to 1
K,10	condk(3)	Set the probe K constant to 10
CAL,CLEAR	condr(0,1)	clear calibration
CAL,DRY	condr(0,2)	Performs dry calibration
CAL,?	condr(0,3)	Query calibration
CAL,ONE,x	condr(1,x)	Performs single point calibration
CAL,LOW,x	condr(2,x)	Performs low point calibration
CAL,HIGH,x	condr(3,x)	Performs high point calibration
X	condx()	Factory reset
STATUS	condst()	Retrieve status information
SLEEP	condsleep()	Enter low power sleep mode
I	condi()	Query device information



## D PLUGIN LAYOUT : JAVASCRIPT

### D.1 file index.js (server side)

```
1 function plugin(name, deps) {  
2  
3     deps.io.sockets.on('connection', function (socket) {  
4         //websockets  
5     });  
6 };  
7 module.exports = plugin;
```

### D.2 file plugin.js (browser side)

```
1 (function (window, $, undefined) {  
2     'use strict';  
3  
4     var plugin;  
5  
6     plugin = function rgb(cockpit) {  
7         console.log("Loading plugin in the browser");  
8         this.cockpit = cockpit;  
9  
10        // Add required UI elements  
11        this.listen();  
12    };  
13  
14  
15    //This pattern will hook events in the cockpit and pull them all back  
16    //so that the reference to this instance is available for further processing  
17    rgb.prototype.listen = function listen() {  
18        var self = this;  
19  
20    };  
21  
22    window.Cockpit.plugins.push(plugin);  
23 }(window, jQuery));
```



## E PLUGIN LAYOUT : C++

### E.1 file plugin.h

```

1  #ifndef __PLUGIN_H_
2  #define __PLUGIN_H_
3  #include <Arduino.h>
4  #include "Device.h"
5  #include "Pin.h"
6  #include "AConfig.h"
7  #include "Timer.h"
8
9  //This must be defined with the right pin. It is set in the cape and controlboard header files.
10 // #define LIGHTS_PIN 5
11 #if (HAS_STD_CAPE)
12     #include "Cape.h"
13 #endif
14
15 #if (HAS_OROV_CONTROLLERBOARD_25)
16     #include "controllerboard25.h"
17 #endif
18
19 class Plugin : public Device {
20     private:
21
22     public:
23         Plugin();
24         void device_setup();
25         void device_loop(Command cmd);
26
27 };
28 #endif

```

### E.2 file plugin.cpp

```

1  #include "AConfig.h"
2  #if (HAS_PLUGIN_SENSOR)
3  #include "Device.h"
4  #include "Pin.h"
5  #include "plugin.h"
6  #include "Settings.h"
7  #include <Arduino.h>
8
9  Plugin::Plugin() : Device() {
10 }
11
12 void Plugin::device_setup() {
13
14     Settings::capability_bitarray |= (1 << PLUGIN_CAPABLE);
15
16     //initialisation code
17
18 }
19
20 void Plugin::device_loop(Command command) {
21
22     //loop code
23 }
24
25 #endif

```





## F OCTAVE SCRIPT : CONDUCTIVITY

```

1 y1 = [1438,1410,1426,1420,1419,1417,1416,1433 ];
2 y2 = [650,654,653,632,651,651,646,630];
3 y3 = [1451,1418,1412,1414,1413,1411,1414,1415];
4 y4 = [769,753,761,749,749,748,749,750];
5
6 x = [1:8];
7
8 M1 = ones(1,8) * mean(y1);
9 M2 = ones(1,8) * mean(y2);
10 M3 = ones(1,8) * mean(y3);
11 M4 = ones(1,8) * mean(y4);
12
13 S1 = ones(1,8) * std(y1);
14 S2 = ones(1,8) * std(y2);
15 S3 = ones(1,8) * std(y3);
16 S4 = ones(1,8) * std(y4);
17
18 figure(1);
19 plot(x, y1, 'o', x, M1, '- ', x, M1 + S1, '--r', x, M1 - S1, '--r');
20 legend('sample value', 'mean', 'standard deviation');
21 ylabel('conductivity [us/cm]');
22 xlabel('samples');
23 title('HI 8733 conductivity sensor');
24
25 figure(2);
26 plot(x, y2, 'o', x, M2, '- ', x, M2 + S2, '--r', x, M2 - S2, '--r');
27 legend('sample value', 'mean', 'standard deviation');
28 ylabel('TDS [ppm]');
29 xlabel('samples');
30 title('HI 8734 TDS sensor');
31
32 figure(3);
33 plot(x, y3, 'o', x, M3, '- ', x, M3 + S3, '--r', x, M3 - S3, '--r');
34 legend('sample value', 'mean', 'standard deviation');
35 ylabel('conductivity [us/cm]');
36 xlabel('samples');
37 title('Atlas Scientific conductivity sensor');
38
39 figure(4);
40 plot(x, y4, 'o', x, M4, '- ', x, M4 + S4, '--r', x, M4 - S4, '--r');
41 legend('sample value', 'mean', 'standard deviation');
42 ylabel('TDS [ppm]');
43 xlabel('samples');
44 title('Atlas Scientific sensor (TDS value)');

```



## G OCTAVE SCRIPT : FILTER

```

1  figure(1);
2  plot(x);
3  ylabel('input voltage [V]');
4  xlabel('samples');
5  title('unfiltered values');
6
7
8
9  b = ones(1,50)/50;
10 y = filter(b, 1, x);
11
12
13 figure(2);
14 plot(y);
15 ylabel('input voltage [V]');
16 xlabel('samples');
17 title('values after moving average');
18
19
20 figure(3);
21 plot(y);
22 axis([50 140 1.8 1.9]);
23 ylabel('input voltage [V]');
24 xlabel('samples');
25 title('values after moving average');
26
27 M1 = ones(1,140) * mean(x(50:end));
28 M2 = ones(1,140) * mean(y(50:end));
29
30
31 S1 = ones(1,140) * std(x(50:end));
32 S2 = ones(1,140) * std(y(50:end));
33 disp("mean is:");
34 disp(M1(1));
35 disp("std deviation input is:");
36 disp(S1(1));
37 disp("std deviation output is:");
38 disp(S2(1));
39
40 figure(4);
41 plot(x, 'o', M1, '-', M1 + S1, '--r', M1 - S1, '--r');
42 axis([50 140 1.7 2.2]);
43 legend('sample value', 'mean', 'standard deviation');
44 ylabel('input volatage [V]');
45 xlabel('samples');
46 title('unfiltered dataset');
47
48 figure(5);
49 plot(y, 'o', M2, '-', M2 + S2, '--r', M2 - S2, '--r');
50 axis([50 140 1.8 1.9]);
51 legend('sample value', 'mean', 'standard deviation');
52 ylabel('input volatage [V]');
53 xlabel('samples');
54 title('filtered dataset');
55
56 figure(6);
57 plot(x, 'o', y, 'x', M1, '-');
58 axis([50 140 1.7 2.2]);
59 legend('unfiltered value', 'filtered value', 'mean of unfiltered values');
60 ylabel('input volatage [V]');
61 xlabel('samples');
62 title('unfiltered vs filtred values');

```







## I BILL OF MATERIAL

### I.1 OpenROV

Product	Provider	Price	Quantity
OpenROV v2.6 kit	OpenROV	849\$	1
OpenROV IMU/Depth Module	OpenROV	80\$	1

### I.2 Conductivity

product	manufacturer	price	quantity
Conductivity sensor K = 0.1 ENV-40-EC-K0.1	Atlas Scientific	120\$	1
EZO Conductivity circuit EZO-EC	Atlas Scientific	43\$	1
Single Circuit Carrier board SCCB	Atlas Scientific	9\$	1
Conductivity Calibration solutions K=0.1 set	Atlas Scientific	12.75\$	1

The probe, the circuit and the calibration solutions have been purchased in a set for a preferential price of 171.95\$

### I.3 Temperature circuit

product	Reseller	price	quantity
Thermistor NTC 10K ohm 615-1075-ND	Digikey	5.61\$	1
MCP1541-I/TO-ND Voltage reference	Digikey	0.92\$	1
LT1006CN8#PBF-ND Operation Amplifier	Digikey	3.12\$	1
V2018-ND PC Board	Digikey	5.75\$	1

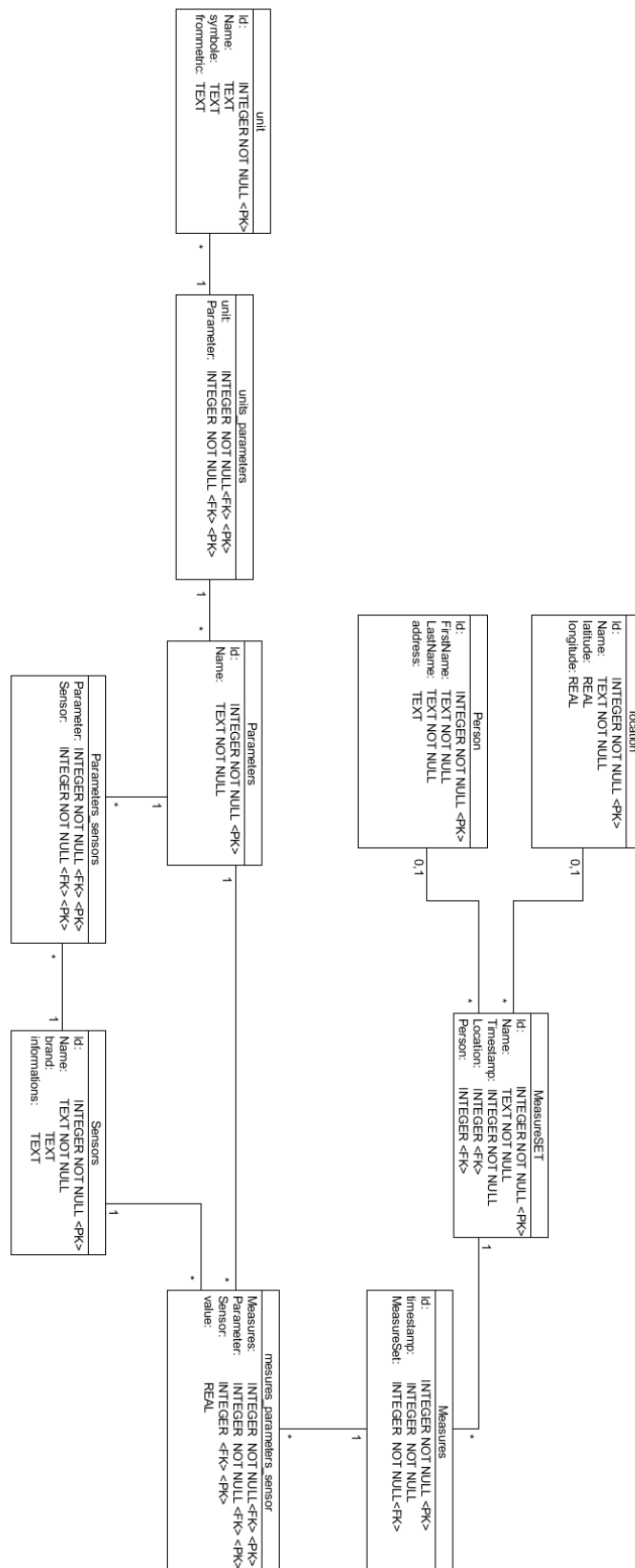
### I.4 Mechanical parts

product	product number	reseller	price	quantity
Sure-Grip Cushioned Loop Clamp, 300 Series Stainless Steel, Silicone Cushion, for 1-1/4" OD (pack of 5)	3177T77	McMaster-Carr	12\$	2
Metric type 316 Stainless Steel Hex Nut, M5 Size (pack of 50)	94150A340	McMaster-Carr	4\$	1
Metric type 316 Stainless Steel Lock washer, M5 Size (pack of 50)	94150A340	McMaster-Carr	4.\$	1
Weather-Resistant Hook and Loop, 2" X 5'	95005K831	McMaster-Carr	16\$	1





## J DATABASE RELATIONAL SCHEMA





## K ROVER ISSUES

## Communication Issue

The problem occurred after the accident in which the 2 communication wires were torn apart and concerns the communication between the computer and the BeagleBone black.

This communication is made the following way:

A twisted pair cable connects the computer and the Topside adapter box as shown in the figure 1. This adapter makes an adaptation between the twisted pair cable and the 2 wires that are connected to the rover. There are 2 circuits inside this box, one is provided by OpenROV as the **Topside interface Board v2.6** and is shown in the figure 2. The other, the green one is a part of a **Tenda P200 powerline adapter**, and can be shown in the figure 3.



Figure 1: Topside adapter

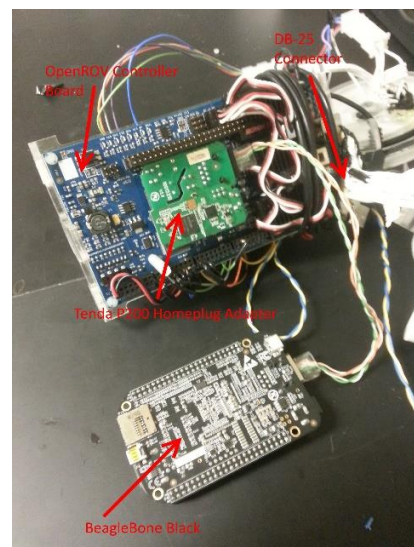
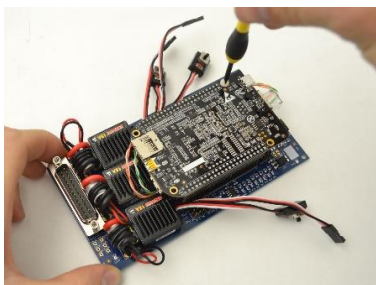
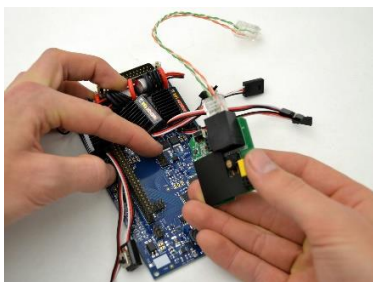


Figure 3: OpenROV Topside interface Board v2.6



Figure 2: Tenda P200 circuit

On the rover side, the 2 wires are connected to the **OpenROV Controller Board** through a DB-25 Connector. It is another **Tenda P200 circuit** (figure 2), plugged on the OpenROV Controller Board that converts the 2 wires communication to a twisted pair cable connected to the **BeagleBone Black**. The figures below shows how everything is set up on the rover.



The symptoms of this problem are:

- The led "HP" (Homeplug) on the *OpenROV Controller board* should be ON and is not.
- The Homeplug led on the *OpenROV Topside interface board* should be ON and is not.
- The communication between the rover and the computer is most of the time not possible (but could work a few times, even with the homeplug leds off).

There are several possible reasons for this problem:

- The wires could be broken in another place. It has been verified by a measurement, which indicates a proper connection.
- One possibility is that water entered into the wires and traveled to the DB-25 connector and shorted them together. Measures have been taken to verify this and no shortages were measured.
- This water could also have shorted the communication wires to the rover power supply, which could have damaged one of the circuits.
- Some water was inside the electronic tube after the test. This water could also have damaged one of the circuits.

If the **BeagleBone black** is connected directly to the computer by a twisted pair cable, the system is working, which means that only the connection causes trouble, and that the BeagleBone black is perfectly working.

It is difficult to identify exactly where the problem is. It could be either one of the **Tenda P200 circuits**, the **OpenROV topside interface** or even the **OpenROV controller board**.

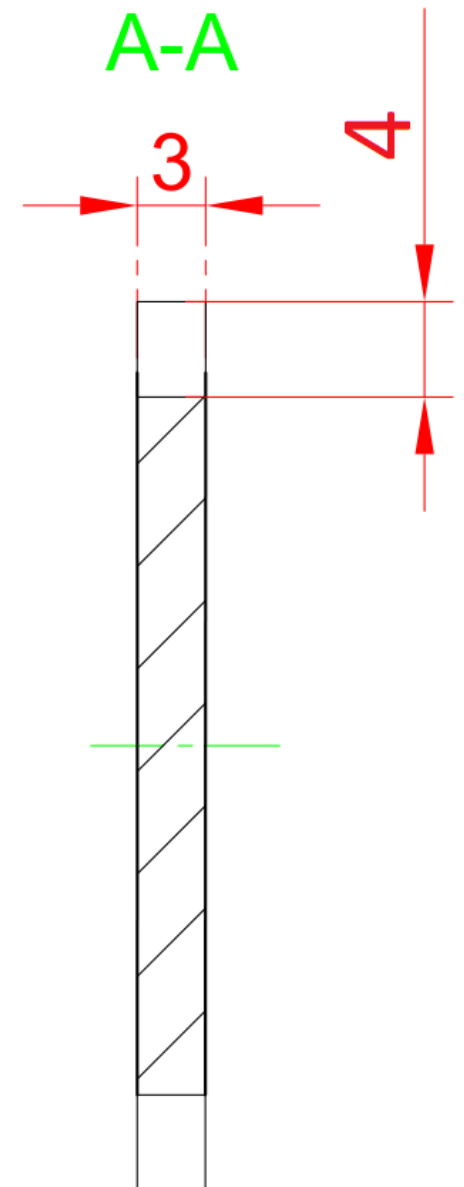
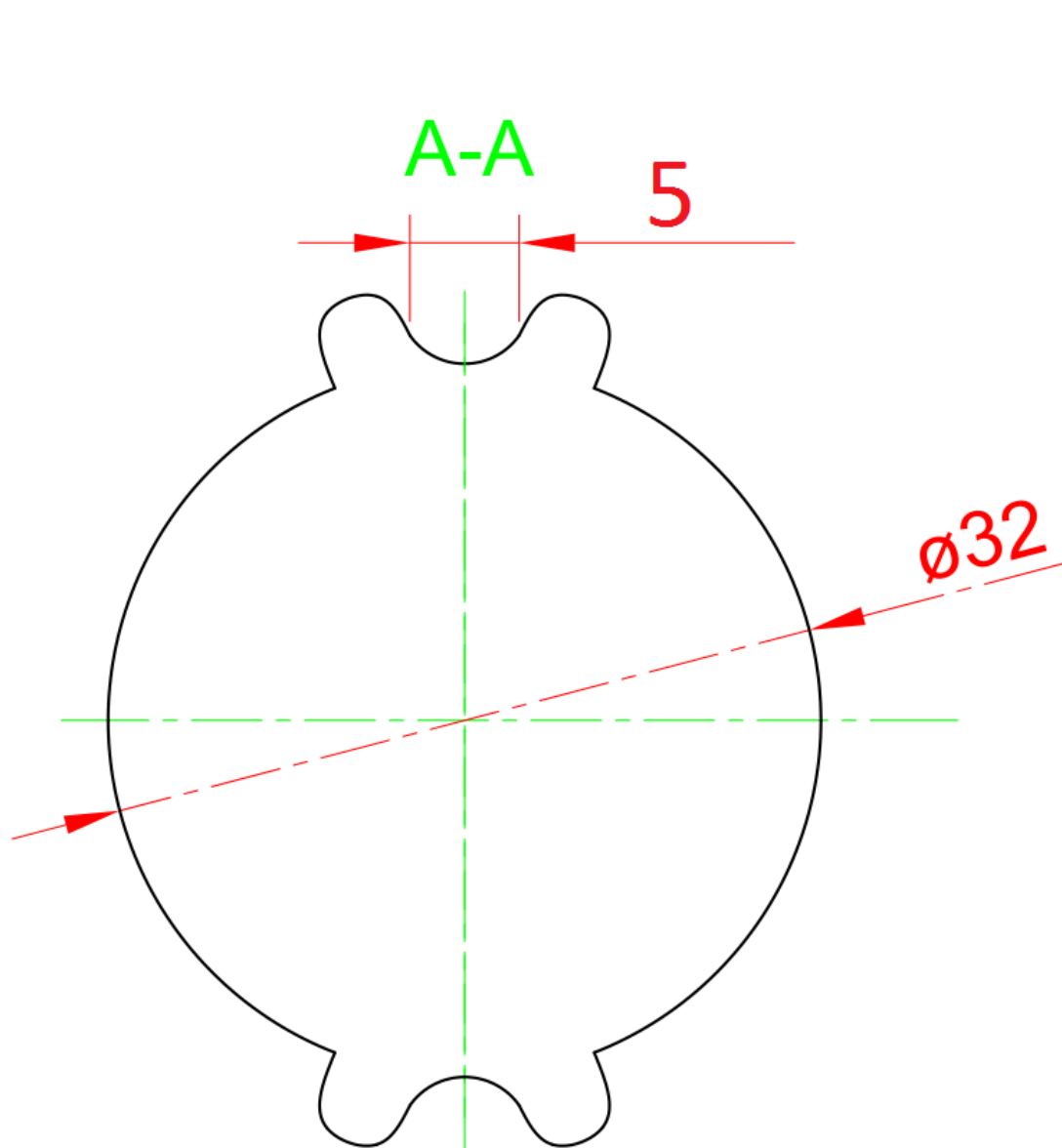
The Tenda P200 circuits could be tested separately of the system between 2 computers in order to know if they are faulty. It will however be required to find how those circuit can be powered and directly connected.

If the Tenda P200 are working this would narrow the problem to the **OpenROV topside interface** and the **OpenROV controller board**.

It seems also reasonable to remake the DB-25 connector before changing any circuit, since water that could have traveled through the wires might be trapped in the insulation I added on the connector.

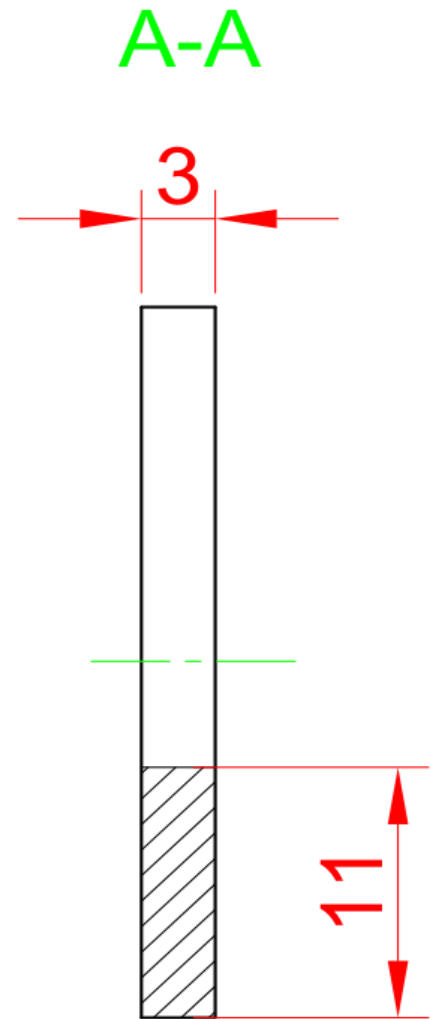
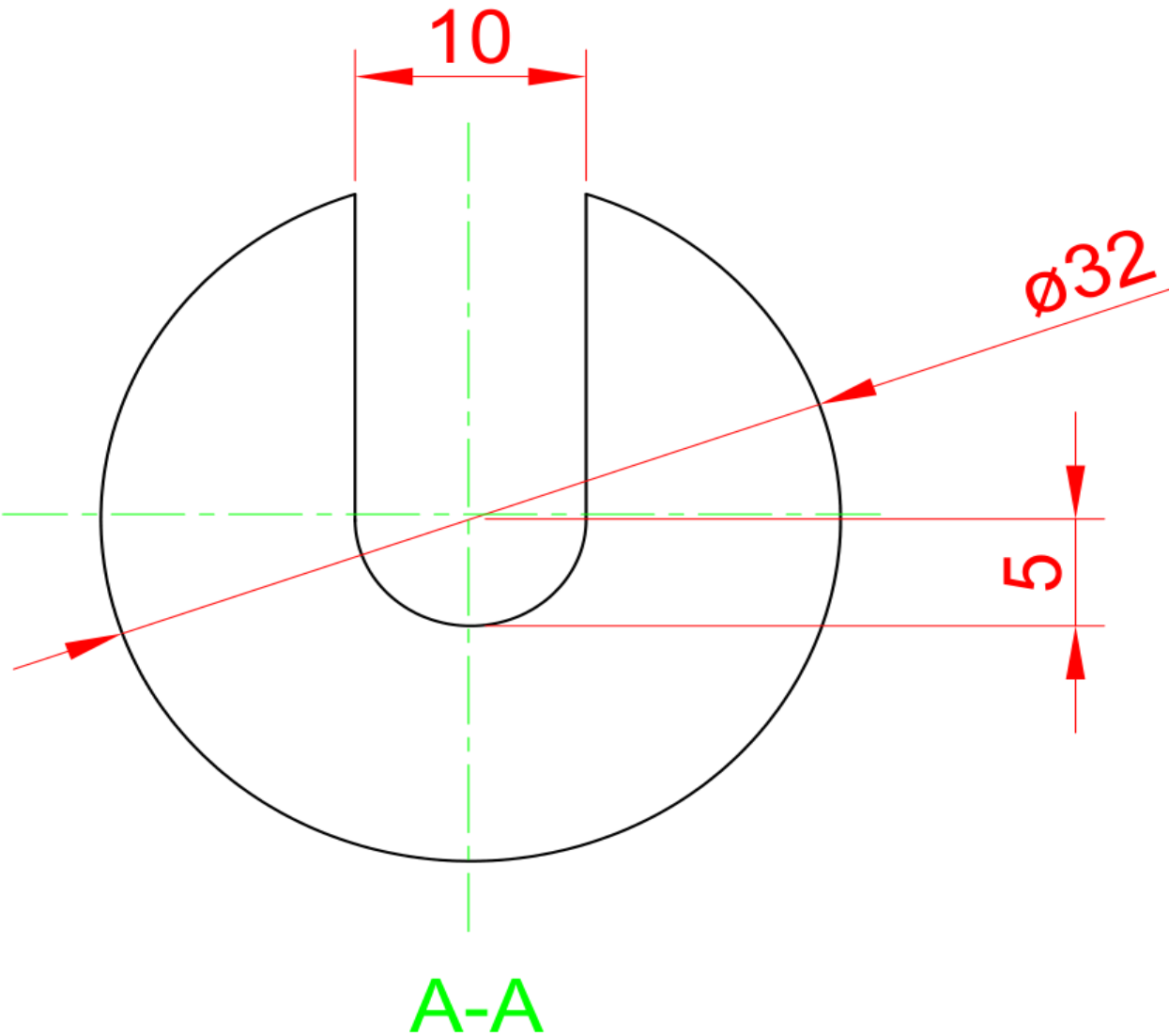


## L ENDCAP DESIGN: ACRYLIC SHEET PIECES

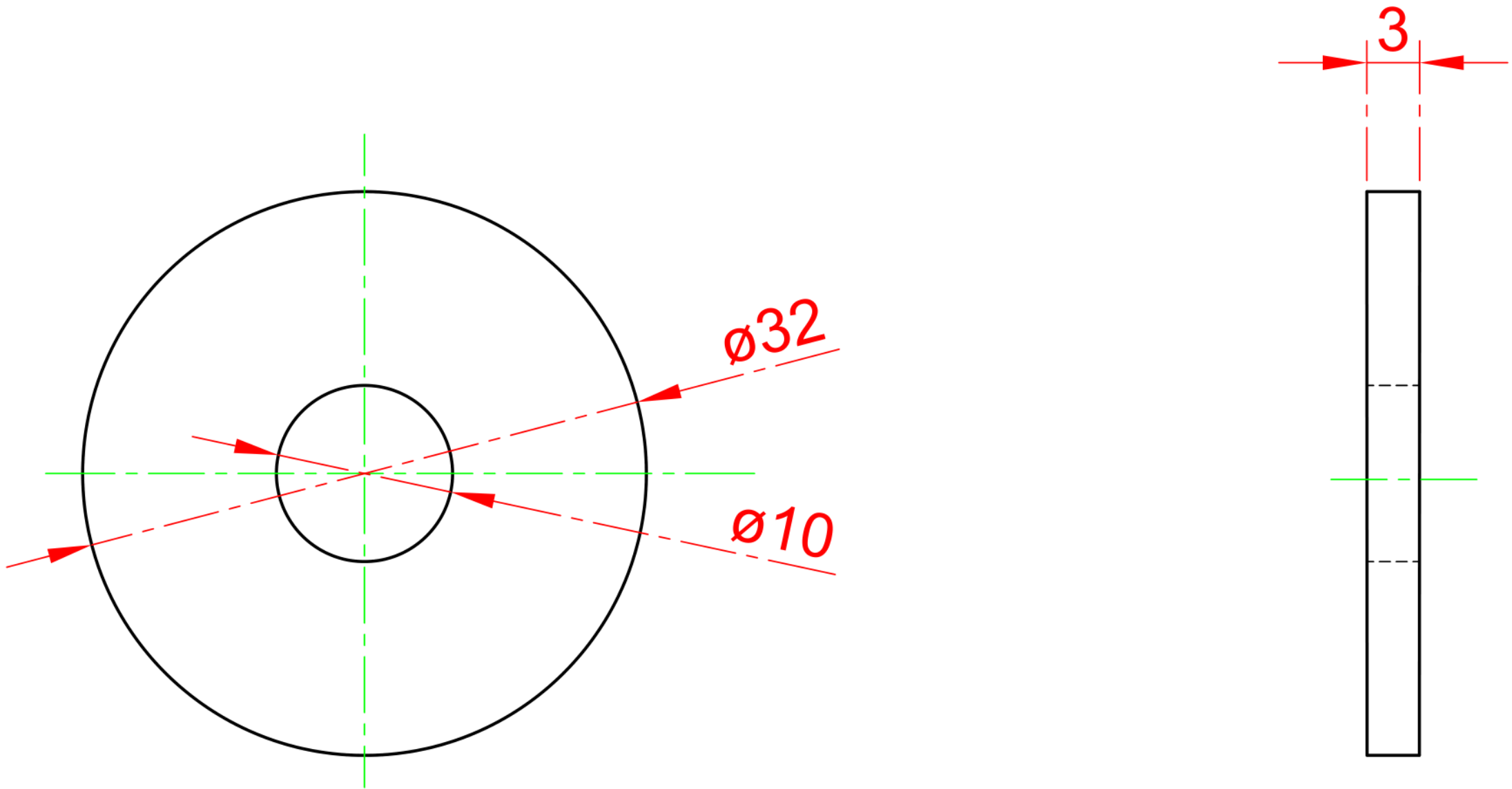


Part Name : Exterior	Part Number : 1
Date: 10.09.2014	Author : Yannick Baehler
Comment: Endcap exterior, fixation for security rubber	

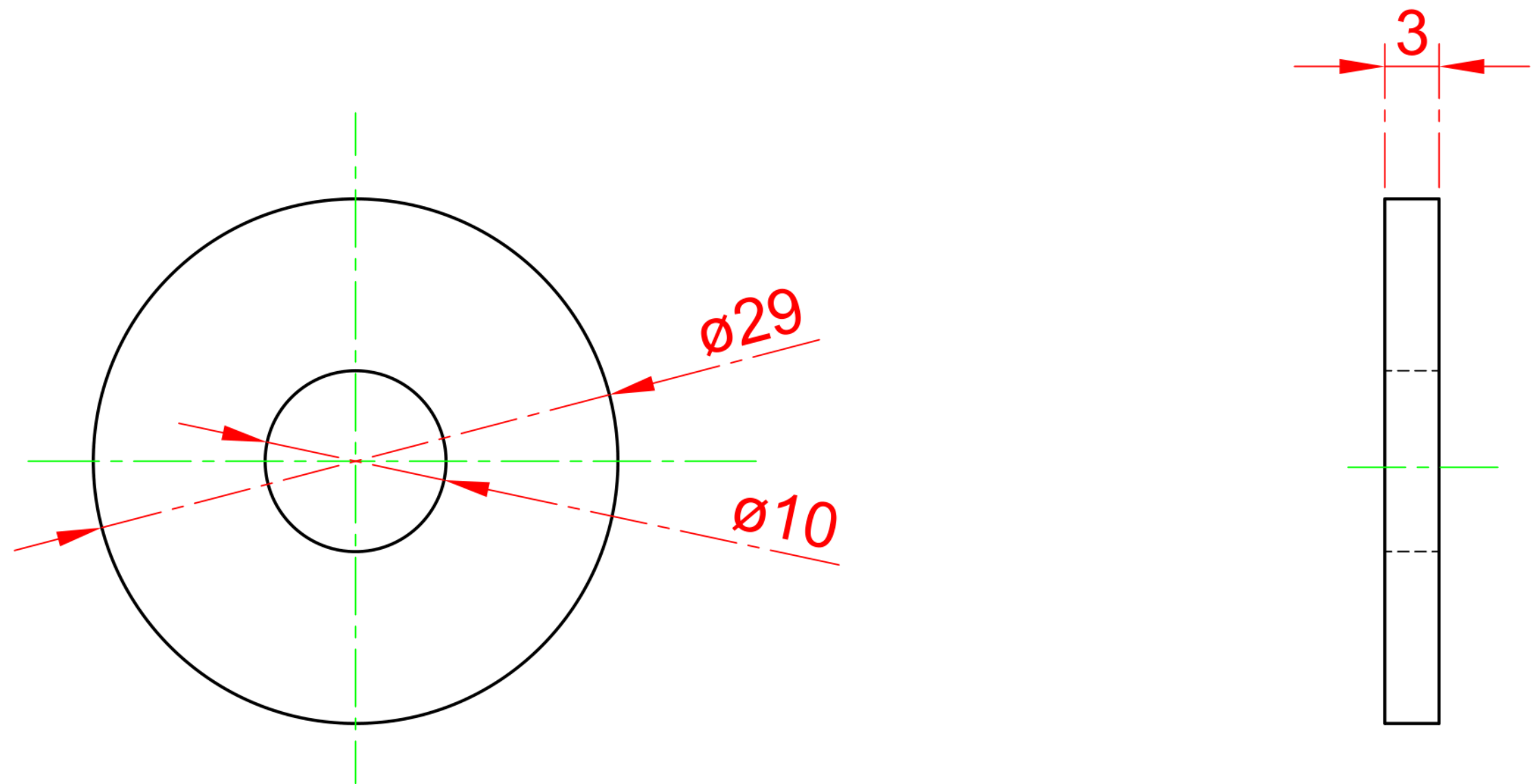




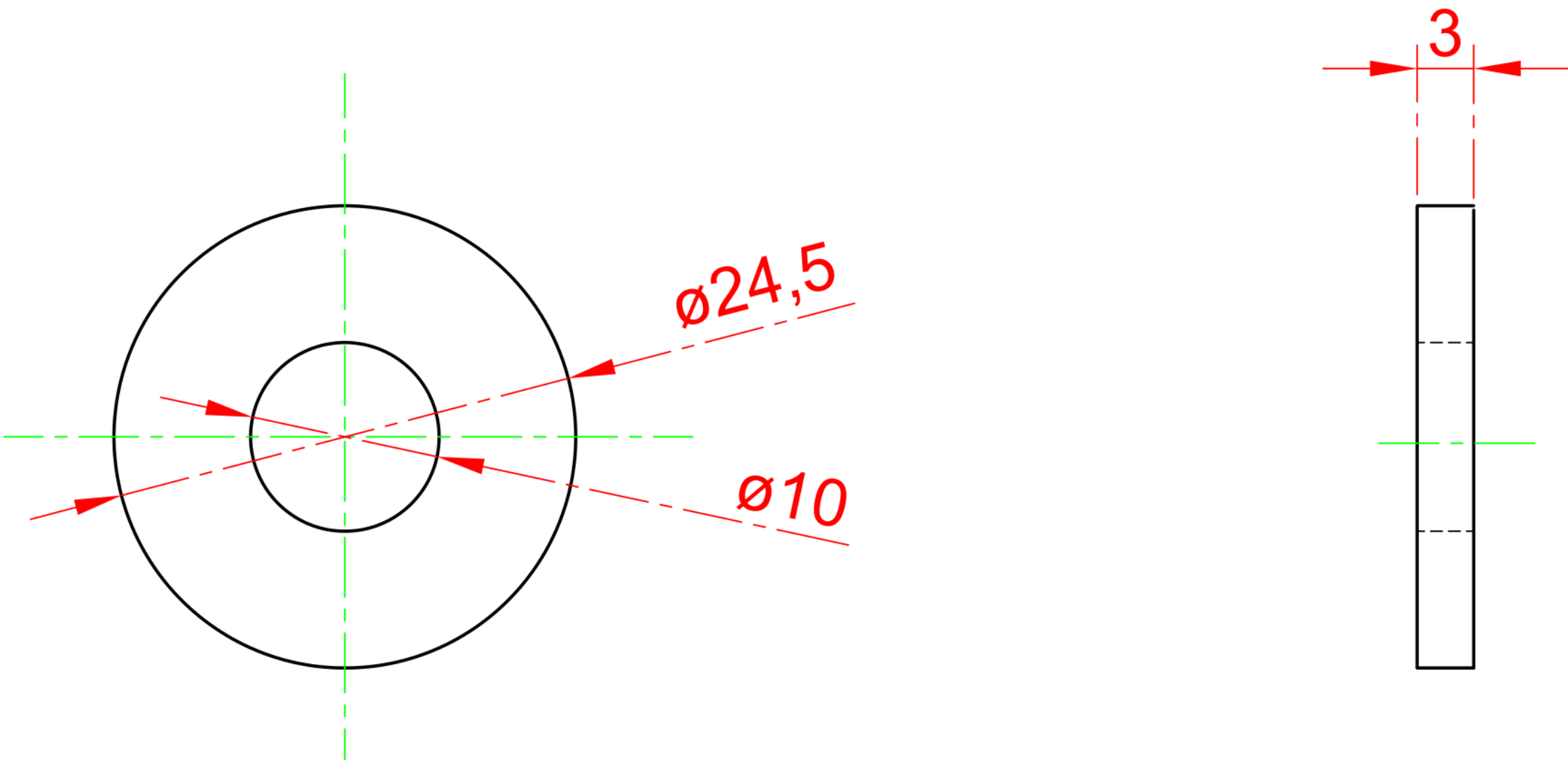
Part Name : Wires_out	Part Number : 2
Date: 10.09.2014	Author : Yannick Baehler
Comment: Exit point for wires	



Part Name : Hole_Exterior	Part Number : 3
Date: 10.09.2014	Author : Yannick Baehler
Comment: Wires pathway	

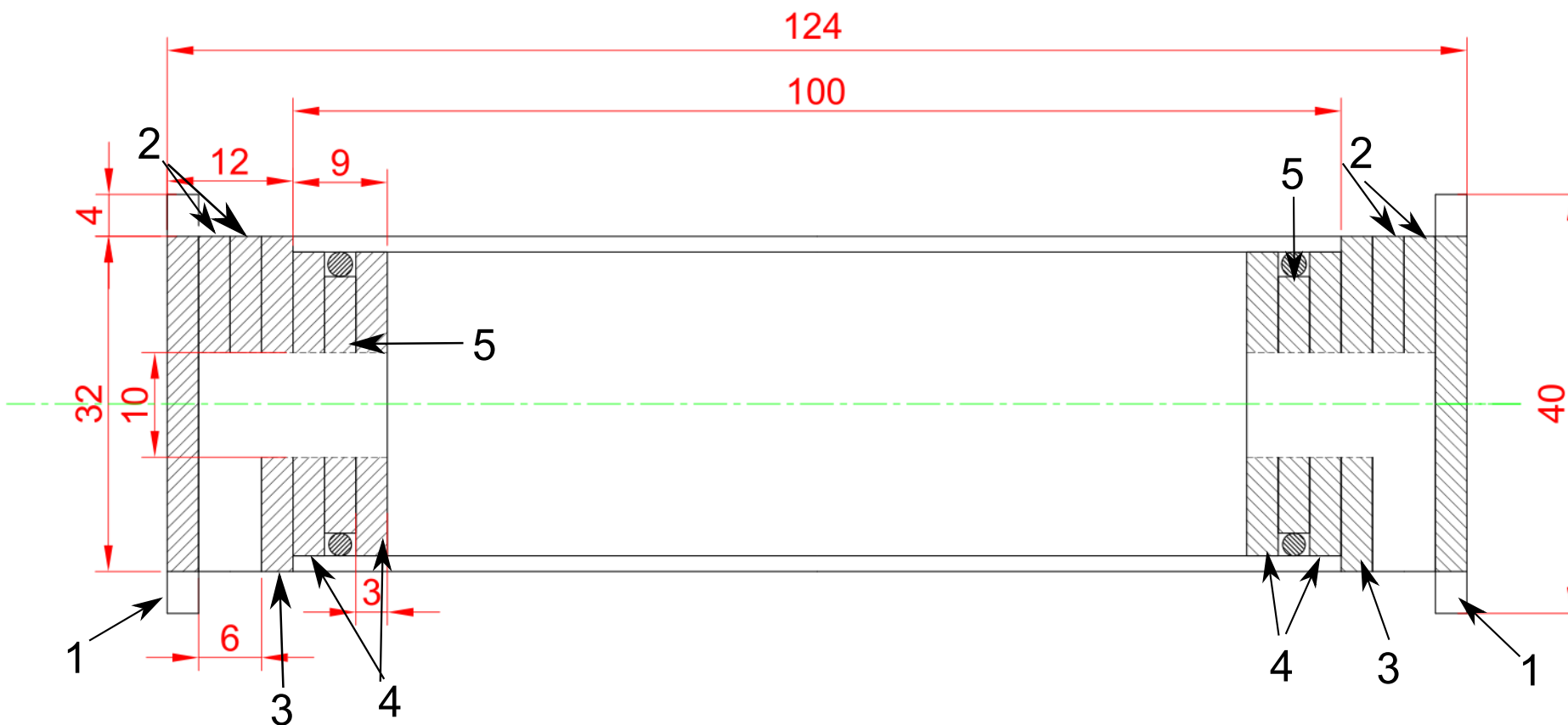


Part Name : o-ring large	Part Number : 4
Date: 10.09.2014	Author : Yannick Baehler
Comment: Wires pathway and exterior of the o-ring socket	



Part Name : o-ring small	Part Number : 5
Date: 10.09.2014	Author : Yannick Baehler
Comment: Wires pathway and interior of the o-ring socket	

## M ENDCAP DESIGN: ACRYLIC SHEET ASSEMBLY



Parts used:

1. Exterior
2. Wires\_out
3. Hole\_Exterior
4. O-ring large
5. O-ring small

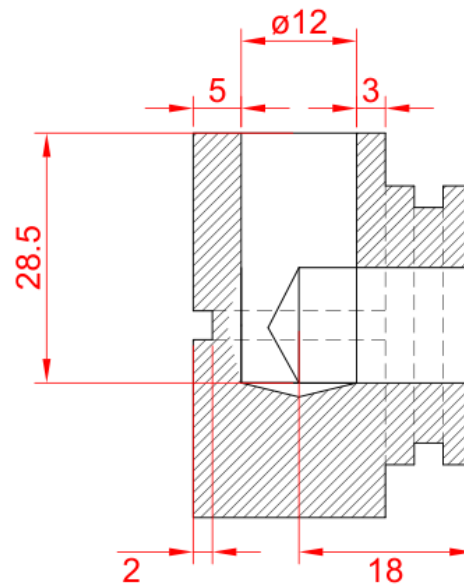
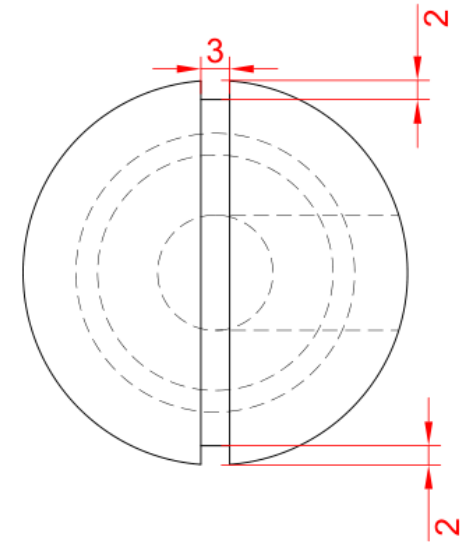
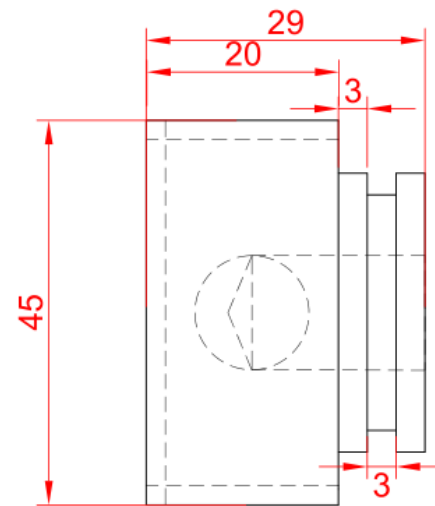
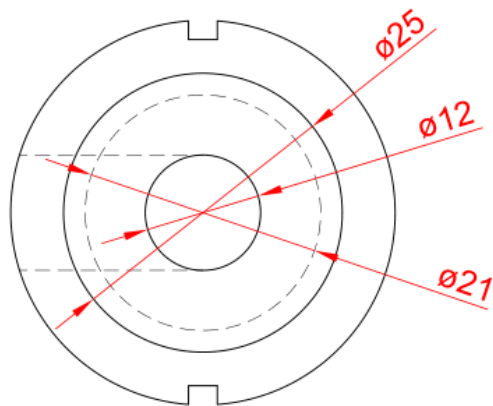
Name : Endcap assembly

Date: 10.09.2014

Author : Yannick Baehler

Comment: Assembly for the endcaps, with tube

## N ENDCAP DESIGN: ACRYLIC BLOCK



Name : Oneblock endcap

Date: 19.09.2014

Author : Yannick Baehler

Comment: Endcap built in one block



## O OPENROV CONTROLLER BOARD SCHEMATICS [13]

# OpenROV Controller 2.6

## Digital I/O Channels

0	BB UART	28	User J3-4
1	BB UART	29	User J3-2
2	User J3-11	30	User J2-16
3	User J3-9	31	User J2-14
4	User J3-7	32	User J2-12
5	User J3-5	33	User J2-10
6	Servo1 J8-1	34	User J2-8
7	Servo2 J8-4	35	User J2-6
8	Servo3 J8-7	36	User J2-4
9	Servo4 J8-10	37	User J2-2
10	Servo5 J8-13	38	N/C
11	Servo6 J8-16	39	N/C
12	PWM4 J1-7/8	40	N/C
13	LED	41	N/C
14	User J2-11	42	N/C
15	User J2-9	43	N/C
16	ESC Power Switch	44	PWM1 J1-1/2
17	N/C	45	PWM2 J1-3/4
18	User J2-7	46	PWM3 J1-5/6
19	User J2-5	47	N/C
20	I2C SDA J1-14	48	N/C
21	I2C SCL J1-12	49	LED
22	User J3-16	50	SPI MISO to BB and ICSP Header
23	User J3-14	51	SPI MOSI to BB and ICSP Header
24	User J3-12	52	SPI SCK to BB and ICSP Header
25	User J3-10	53	SPI SS to BB
26	User J3-8		
27	User J3-6		

## Analog I/O Channels

0	Controller and BB Battery Current
1	ESC 3 Current
2	ESC 2 Current
3	ESC 1 Current
4	Battery Voltage (after protection diodes)
5	Battery 2 Current
6	Battery 1 Current
7	Humidity (optionally populated)
8	Board Temperature
9	User J4-14
10	User J4-12
11	User J4-10
12	User J4-8
13	User J4-6
14	User J4-4
15	User J4-2

## Revision History

Rev	Date	Engr	Changes
2.5	13 Aug 2013	RWH	Initial Prototype
2.5 Rev A	20 Aug 2013	RWH	Prototype Batch. Revised J6 connector orientation. Changed BB node names. Revised C11, C15, and U5 and added C34 to avoid +5V and +3.3V brownout when switching ESCs on. Revised ESC mounting holes. Revised LED colors. Revised silkscreen. Added weak Arduino reset pullup for standalone operation. Added cap slot C35 to hold vehicle on if noisy tether connection. Added bypass points J17 for ESC power switch. Arduino reset function moved from BB pin 13 to BB pin 11.
2.5 Rev B	2 Sep 2013	RWH	First production batch. J6 changed from socket to male board stacker. 4th mounting hole added to board. Revised fiducial locations. Inverted logical case of UART LEDs. BB TX LED moved from Arduino space to BB space. Board ID EEPROM circuitry DNP.
2.6	12 Nov 2013	RWH	Layout changes to accept V2.0 Tenda/MediaLink Homeplug adapter. Additional sockets added to accept V2.0 pin configuration. LEDs added to display Homeplug adapter status. Board ID circuitry modified. Servo output connector split into two. Filters added to ESC current monitors. Test points updated. Silkscreen updated.

## Sheet Finder

1	Cover Page
2	Input / Output
3	Power Switching
4	5V and 3.3V Power
5	Microcontroller (Arduino Mega compatible)
6	PWM and Environment Sensors
7	I2C and BeagleBone Interface
8	Motor Current Sensing and ESC Connections

## OpenROV

This work is released under the Creative Commons Attribution-NonCommercial-Share Alike 3.0 Unported License. All derivative works are to be attributed to OpenROV

## Cover Page

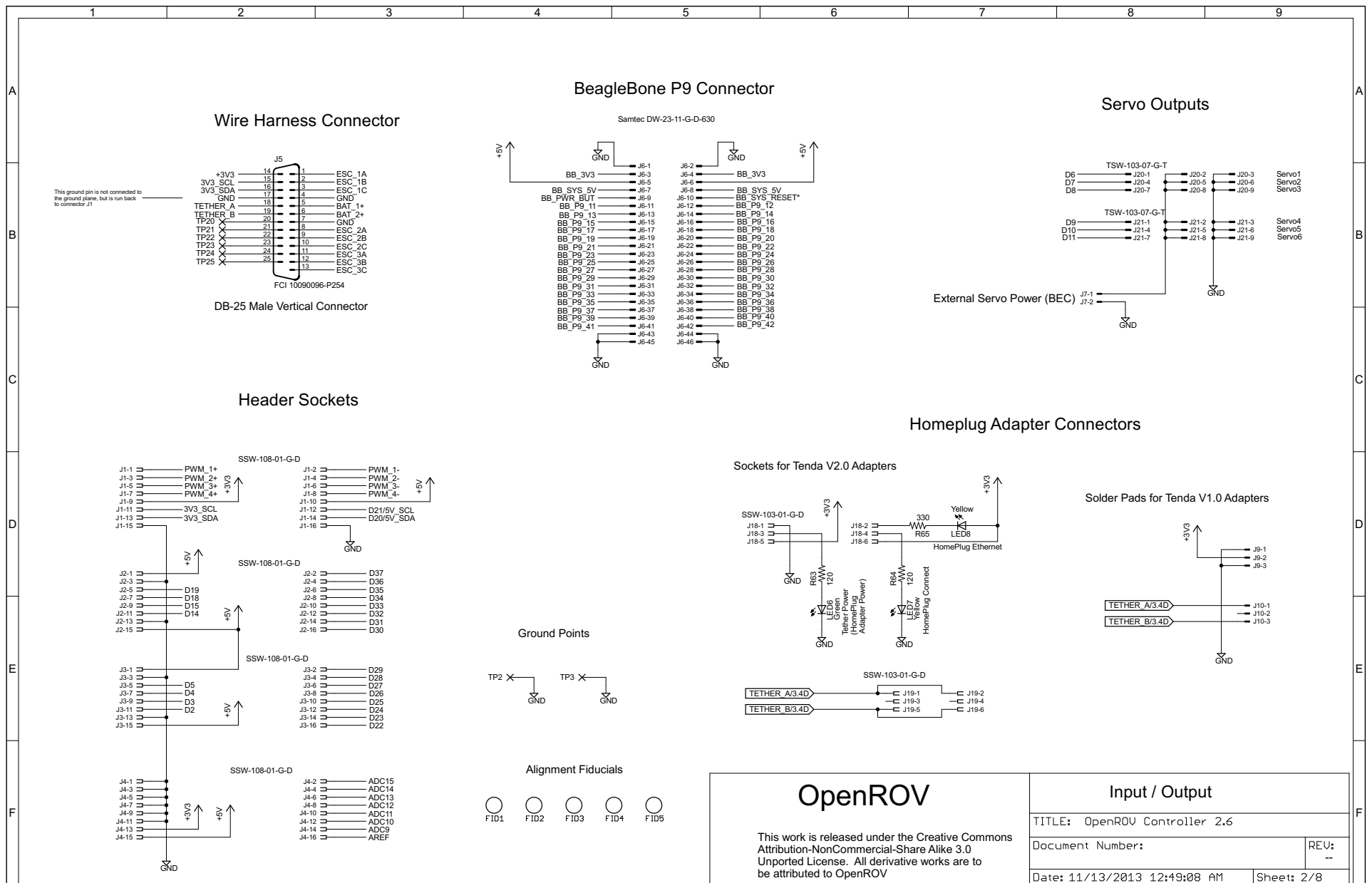
TITLE: OpenROV Controller 2.6

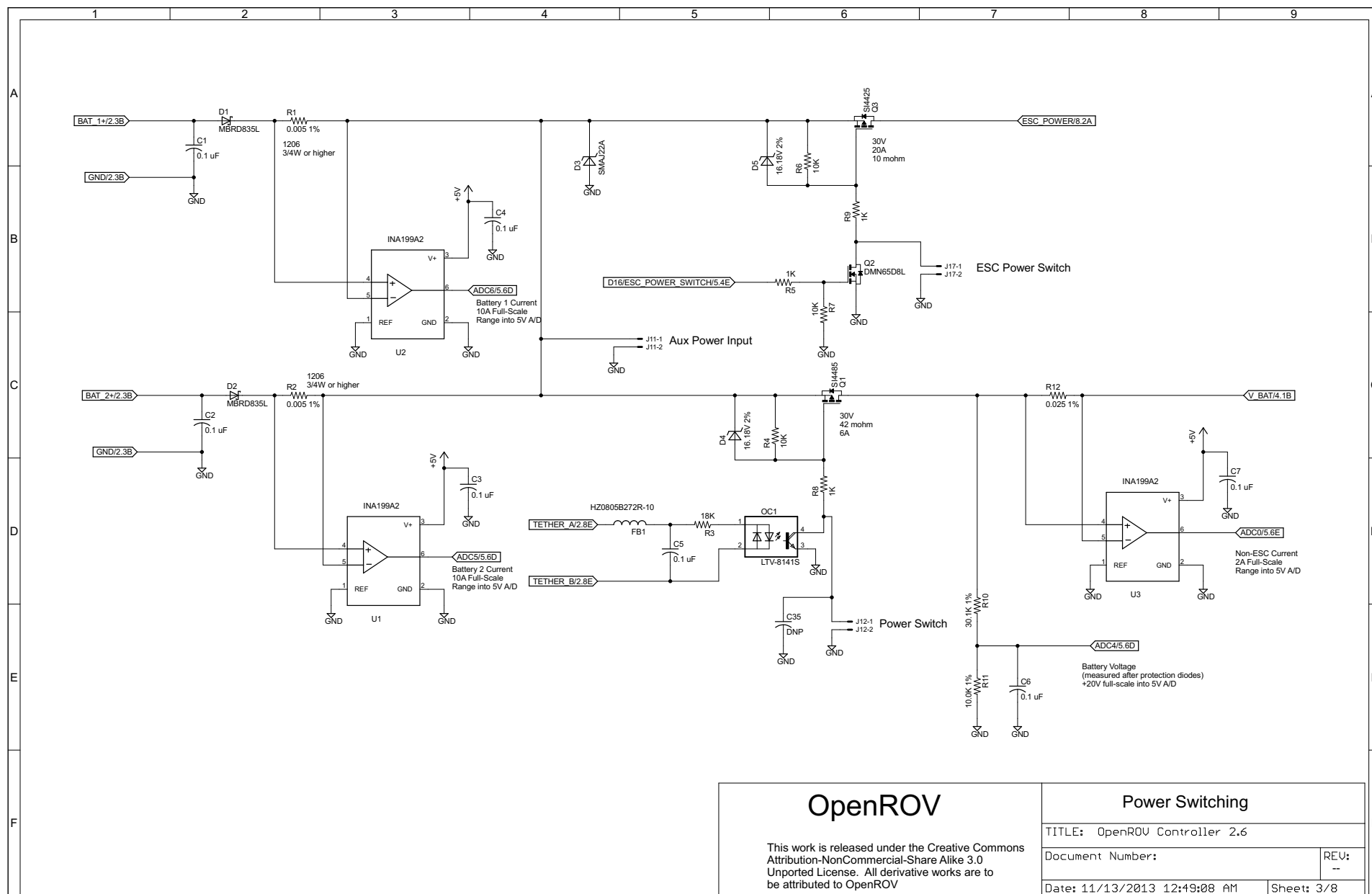
Document Number:

REV:  
--

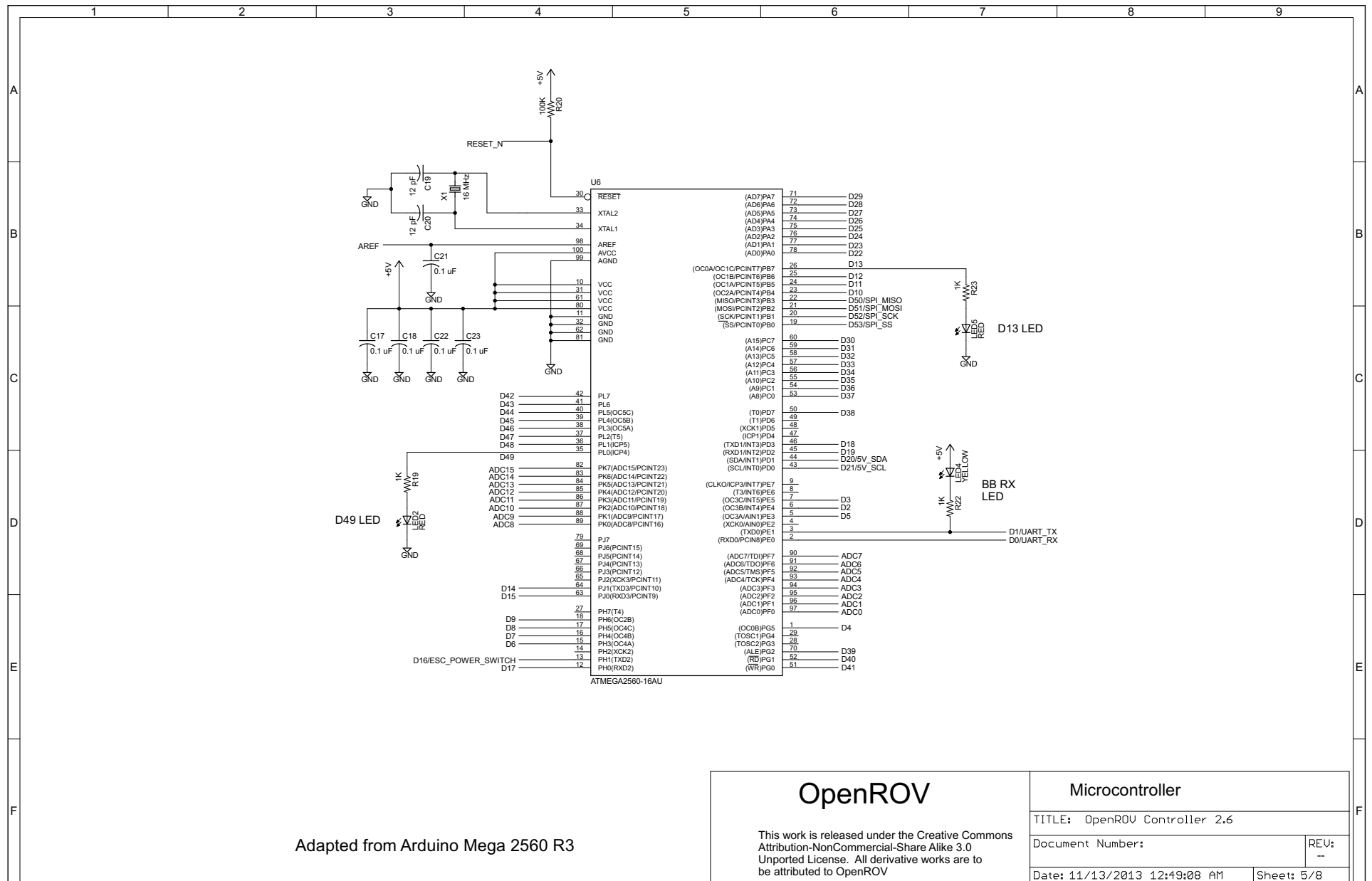
Date: 11/13/2013 12:49:08 AM

Sheet: 1/8









Adapted from Arduino Mega 2560 R3

## OpenROV

This work is released under the Creative Commons Attribution-NonCommercial-Share Alike 3.0 Unported License. All derivative works are to be attributed to OpenROV

## Microcontroller

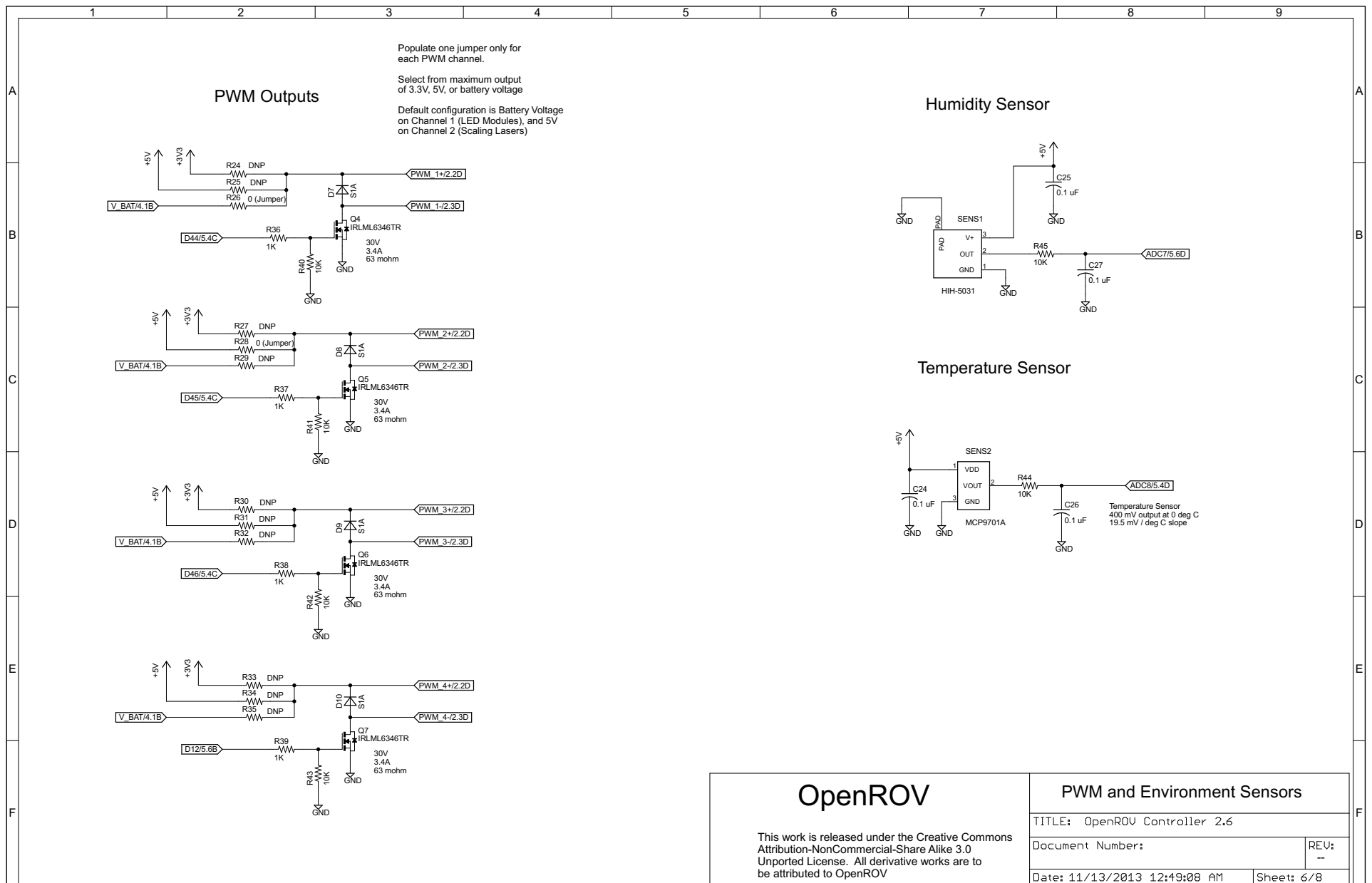
TITLE: OpenROV Controller 2.6

Document Number:

REV:  
--

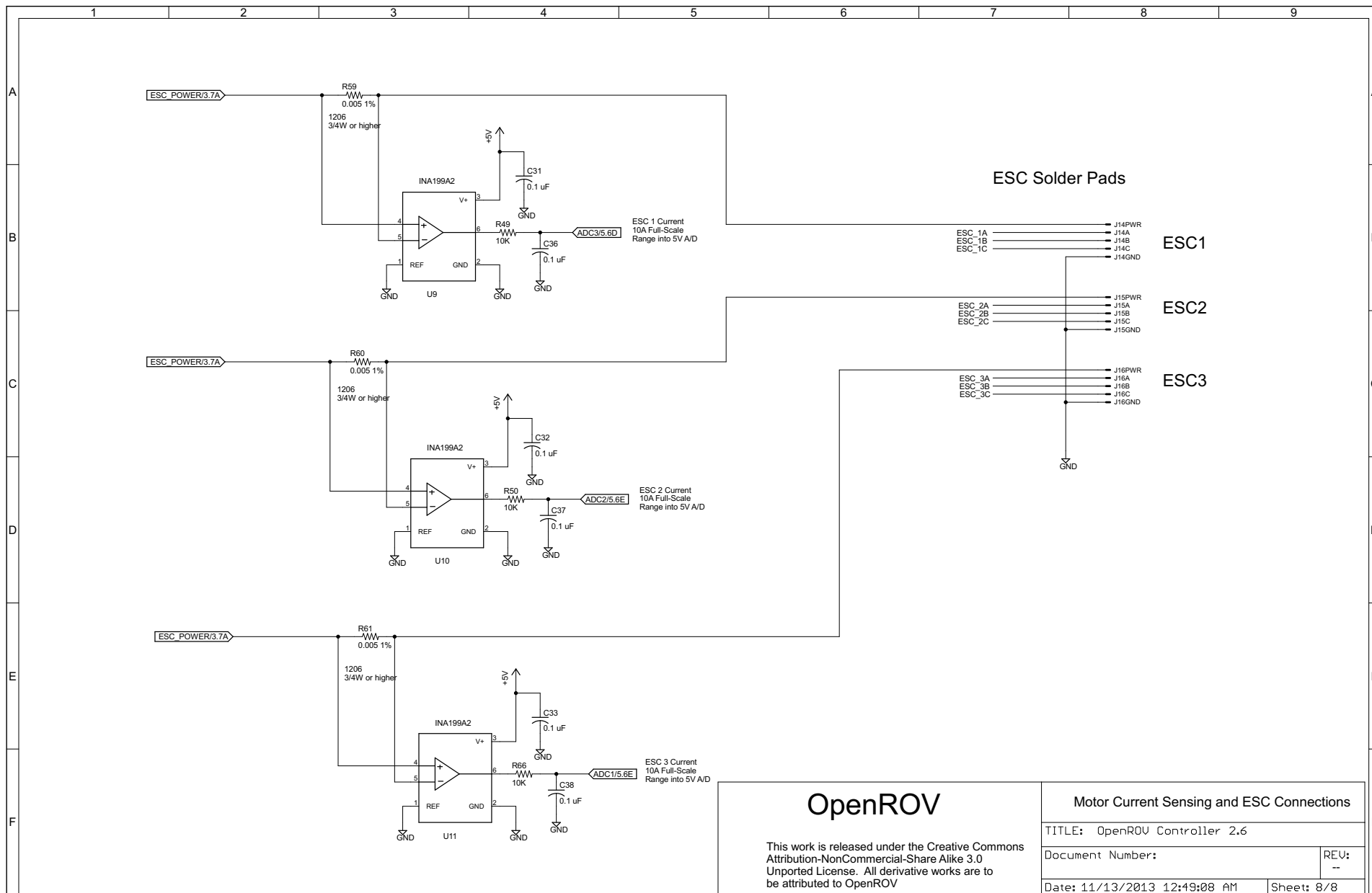
Date: 11/13/2013 12:49:08 AM

Sheet: 5/8











## P BEAGLEBONE BLACK SCHEMATICS [14]

REV	Description	DATE	BY
A4A	Initial production Release.	11/19/2012	GC
A5	On the initial production release the processors were to be found incorrect as supplied by TI. Parts while marked AM3359 were actually AM3352. This revision uses the correct parts.	1/2/2013	GC
A5A	1. Deleted R29-R44 from the LCD lines. 2. Added 47pf capacitors C156-C173 to LCD data lines to ground. 3. Changed schematic revision to A5A. 4. Changed a few footprints after PCB update for above changes. 5. Added access point for the battery function of the TPS65217C. 6. Added Ferrite beads in series with LED power and 5V power rail of the USB host connector. Required to pass FCC/CE testing due to noise emissions on that pin. 7. Added power button to enable sleep, wakeup, power down and power up features on the system. 8. Added Modification to add 100K ohm resistor to ground to prevent crosstalk when serial cable is not plugged in.	2/8/2013	GC
A5B	1. Added 100K pulldown on J1 pin 4 to prevent crosstalk when serial cable is not connected into PCB layout. 2. Changed the LED resistors to 4.75K to lower the brightness.	5/21/2013	GC
A5C	1. Changed R46, R47, R48 to 0 ohms. 2. Changed R45 to 22 Ohms. Change was made due to production failures on some boards due to differences in impedances.	6/12/2013	GC
A6	1. Moved the enable for the VDD_3V3B regulator to VDD_3V3A rail. Change was made to reduce the delay between the ramp up of the 3.3V rails. 2. Added a AND gate to the SYS_RESETr circuitry. There is a small chance that on power up the nRESETrOUT signal on the processor may go high, causing the SYS_RESETrIn signal to go HI before it should. This change reinforces the reset with the PORZn reset signal. 3. Added optional zero ohm resistor to tie GND_OSC0 to system ground.	7/25/2013	GC
A6A	1. Added optional zero ohm resistor to tie GND_OSC1 to system ground. 2. Changed C106 to a 1uF capacitor. 3. Changed C24 to a 2.2uF capacitor. 4. Made R8 installed and R9 not installed.	12/13/2013	GC
B	1.Changed the processor to the AM3358BZCZ100.	1/20/2014	GC
C	1.Increased the eMMC from 2GB to 4GB.	3/21/2014	GC

**This schematic is \*NOT SUPPORTED\* and DOES NOT constitute a reference design. Only "community" support is allowed via resources at BeagleBoard.org/discuss.**

**THERE IS NO WARRANTY FOR THIS DESIGN , TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE DESIGN "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE DESIGN IS WITH YOU. SHOULD THE DESIGN PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.**

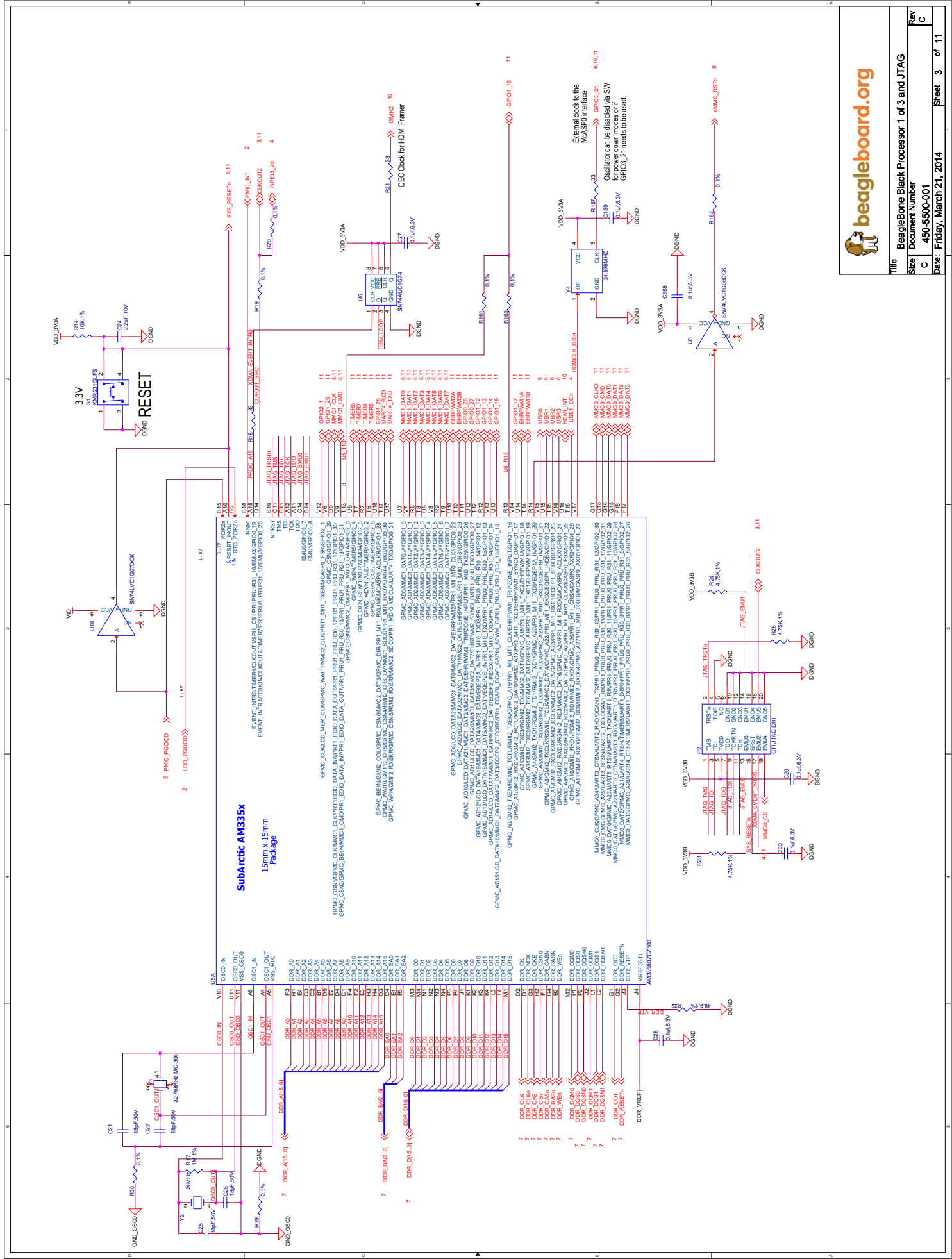
PAGE NO.	SCHEMATIC PAGE
1	COVER PAGE
2	POWER MANAGEMENT
3	PROCESSOR 1 OF 3, JTAG HEADER
4	PROCESSOR 2 OF 3, UAB PORTS
5	PROCESSOR 3 OF 3
6	LED, CONFIGURATION AND BUTTON
7	DDR3 MEMORY
8	eMMC FLASH
9	10/100 ETHERNET
10	HDMI FRAMER
11	EXP CONN, uSD

NOTE: PCB Revision for this board is Rev B6



Title	BeagleBone Black Cover Page		
Size	Document Number	Rev	
B	450-5500-001	C	
Date:	Friday, March 21, 2014	Sheet	1 of 11

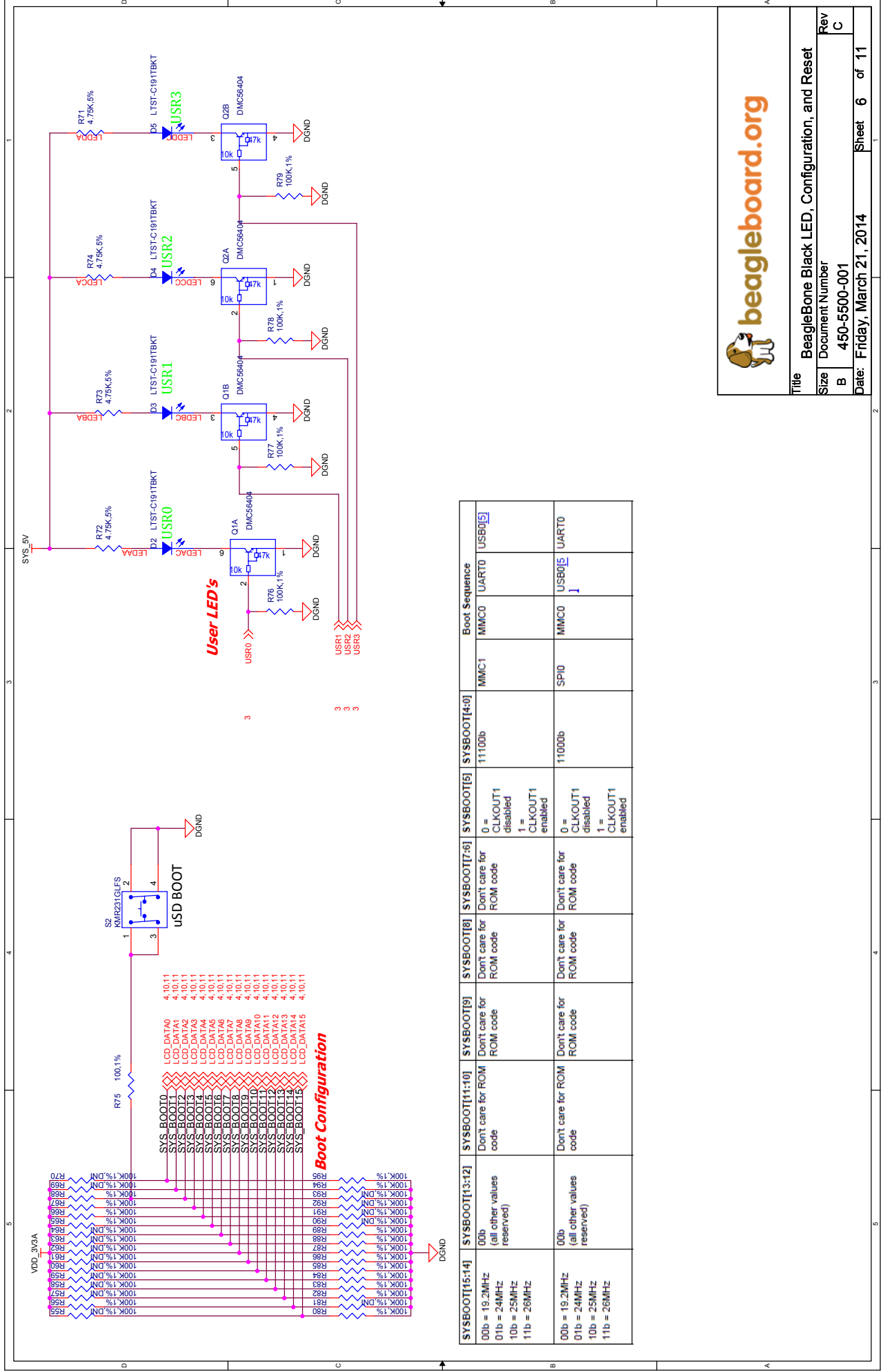




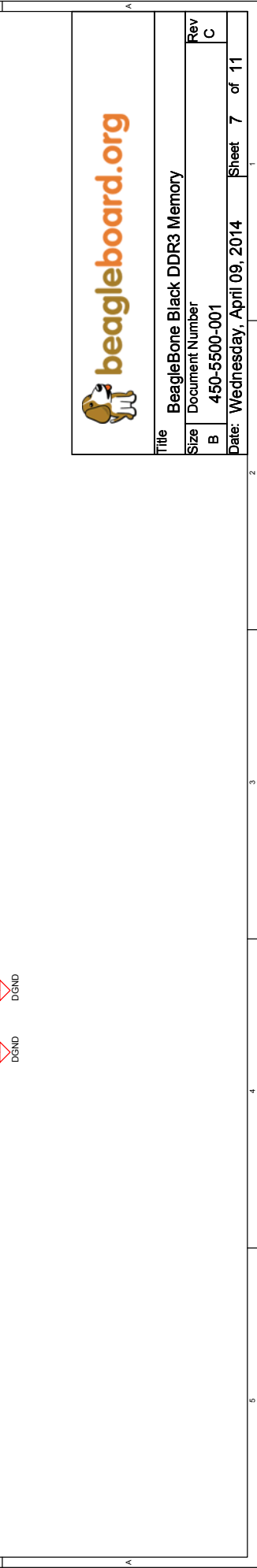








Title		BeagleBone Black LED, Configuration, and Reset	
Size	Document Number	Rev	C
B	450-5500-001		
Date:	Friday, March 21, 2014	Sheet	6 of 11





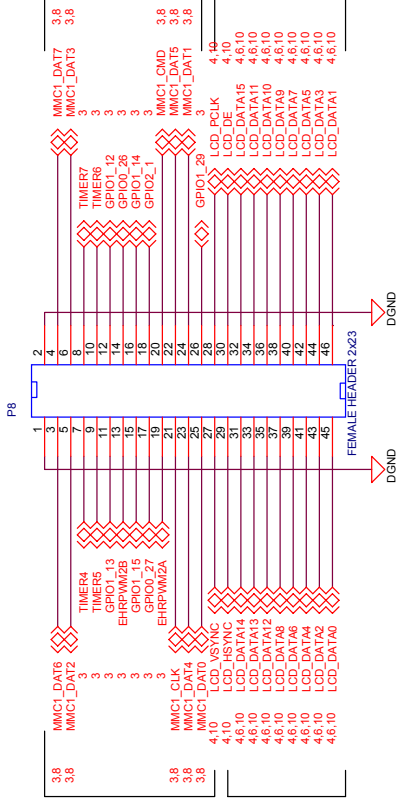
Title		Beagle BoneBlack 4G eMMC	
Size	B	Document Number	Rev C
Date: Friday, March 21, 2014		Sheet 8 of 11	



BeagleBone Black Ethernet		Rev C
Document Number		
Size B	450-5500-001	
Date: Friday, March 21, 2014	Sheet 9	of 11

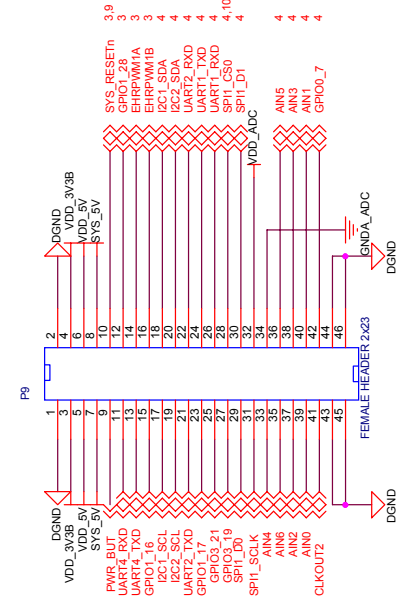


CAUTION: USED ON BOARD

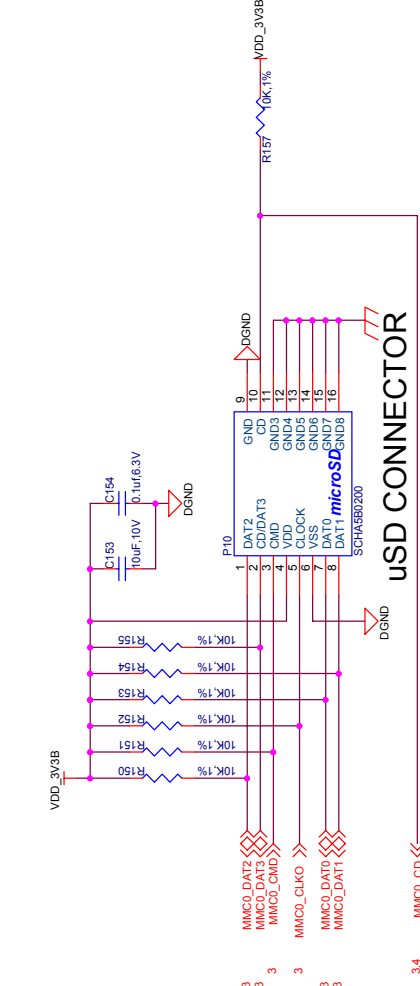


## EXPANSION HEADER

CAUTION: USED ON BOARD

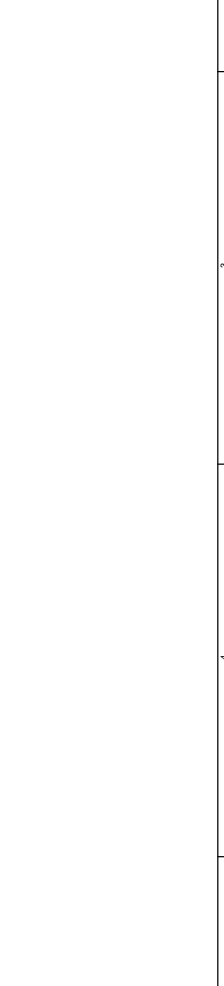


## EXPANSION HEADER



## EXPANSION HEADER

3,4 MMC0\_CD <<<



Title		BeagleBone Black Expansion Headers, uSD and EEPROM
Size	Document Number	Rev C
B	450-5500-001	
Date:	Friday, March 21, 2014	Sheet 11 of 11