M B A R I

A Low Cost, Underwater CO_2 Sensor For Education And Research

Miguel Uzcategui, University Of The District Of Columbia

Mentor: Ken Johnson

Summer 2013

INDEX

Abstract: *Due to the increasing concern on how CO_2 is affecting the seas, and lakes; an underwater CO_2 sensing module was developed to fulfill this task . It works by using Teflon AF tubing to equilibrate the pCO_2 that is underwater. The sensor used was a Telaire 6615 which is a flow through dual channel sensor that communicates to a microcontroller using UART protocol. By putting the device on the Moss Landing harbor, it was determined that the sensor system works in an accurate way, where there was a constant fluctuation of pCO_2 between 470 ppm and 530 ppm which is correlated to the tide. The conclusion of this project, is that a reliable, low cost underwater CO_2 sensing module can be built using Teflon AF as an equilibrator, in the future it is hoped that the sensing module could serve for future small researches or simply for education.*

INTRODUCTION

In the present, humanity have started to realize how important is the chemistry of the water for all the living species on earth. Carbon dioxide plays an important role in the chemistry of our seas, lakes, and rivers (NASA). Many studies are being done to produce better sensors or better algorithms that will detect underwater carbon dioxide levels more efficiently and with more precision (ATABEK) (Ning, Li and Li). The reason of all this studies is simple, to have a better understanding of how carbon dioxide affects life underwater. Some species are more suited to live in an environment with more carbon dioxide than others. Different concentrations of carbon dioxide could trigger different answers to a common stimulus among different species (Damage to Marine Ecosystems as CO2 Emissions Rise). Another reason to study CO_2 , could be on how CO_2 changes affect the metabolism of respiration and photosynthesis in different underwater plants.

The objective of this project is to develop a low cost, easy to build, underwater, and low power consumption carbon dioxide sensor. It is not intended to be a high precision sensing device, but is good enough so that some undergraduate or masters researches could use it to get good data on carbon dioxide concentration that could help them on their respective researches. It could also be used by anybody with low technical skills who just want to conduct a small research for self-benefit.

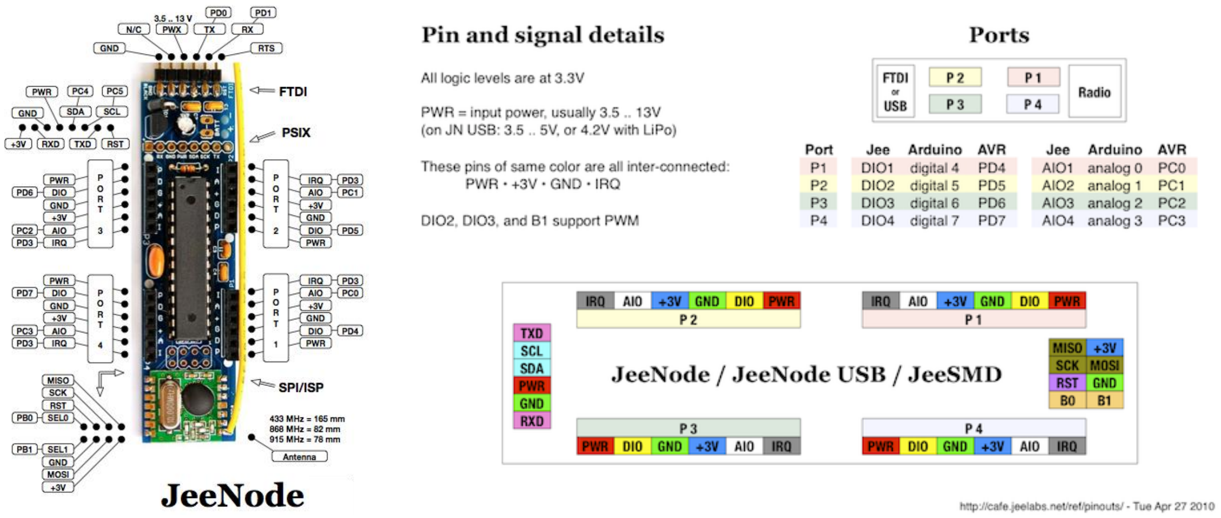
In order to measure CO_2 concentration underwater, the sensor will be introduced on a water tight housing, where a small Teflon AF tubing is going to be in touch with the water and going into the sensor. The Teflon AF tubing serves as an equilibrator of Carbon dioxide, meaning that the same partial pressure of CO_2 that it is underwater will diffuse through the Teflon AF tubing due to its special capabilities (Dasgupta).

The need of developing radio communications to the sensing module was necessary. This was because, as said before, the sensor will be put in a water tight housing; therefore, the less it is taken out of that housing the better. This is to avoid any human error that will damage the housing while opening and closing it, thus getting water in the sensor and the microcontroller when put into the water.

A complete explanation about the CO_2 sensing mote and the Teflon AF tubing will be addressed on the following sections.

COMPONENTS

Microcontroller (Jeenode): The Jeenode is a microcontroller based on the Arduino concept that is designed for low power consumption. It has 4 ports in which you can connect any device using the I2C protocol (Wikipedia, I2C), or also have analog and Digital pins for devices that do not use I2C protocol. The voltage supply is from 3.3 to 17V. The key about the Jeenode is that is designed for low power consumption. The Jeenode consumes 9mA when is in normal operation and 0.8mA (Jeelabs, Jeenode SMD) while it is on Sleepy state; while the Arduino consumes 50 to 500mA while awake, depending on how is running, and 10 mA while it is on sleeping mode (Mitchell).



Pin and signal details

All logic levels are at 3.3V

PWR = input power, usually 3.5 .. 13V
(on JN USB: 3.5 .. 5V, or 4.2V with LiPo)

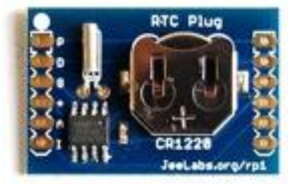
These pins of same color are all inter-connected:
PWR • +3V • GND • IRQ

DIO2, DIO3, and B1 support PWM

Port	Jee	Arduino	AVR	Jee	Arduino	AVR
P1	DIO1	digital 4	PD4	AIO1	analog 0	PC0
P2	DIO2	digital 5	PD5	AIO2	analog 1	PC1
P3	DIO3	digital 6	PD6	AIO3	analog 2	PC2
P4	DIO4	digital 7	PD7	AIO4	analog 3	PC3

Figure 1 (pin out and port description of the Jeenode)

- **RTC plug(Clock):** The clock plug is a Jeenode plug that works with the real time chip DS1340. It has a little battery backup so it keeps working after disconnected from the power source. It runs using an I2C bus running at 3.3V. So it can be connected in any Jeenode port. (Jeelabs, RTC plug)



Port connector:

PWR	DIO	GND	+3V	AIO	IRQ

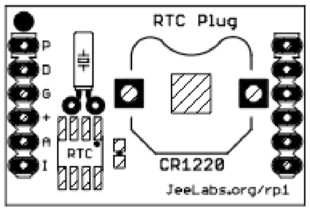


Figure 2 (picture of the RTC plug along with the Pinout description.)

- **Memory plug:** The memory plug is a small board with 1 to 4 EPROM memory chips, each of them have a capacity of 128 Kbytes. It could have up to 512Kbytes of memory; however for this project 2 memory chips are used, which means there is up to 256 Kbytes in memory. The memory plug uses an I2C bus with 3.3V to interface with any device or any of the four Jeenode ports. It is designed on a way that each memory chip is composed of pages and directories. Each memory chip has 1000 pages with a capacity of 256 bytes per page. (Jeelabs, Memory plug)**.** There is no file system for reading or writing; therefore all the readings and writings have to be done manually by the user. The best way to do it, is by setting up a directory on page 0 byte 0 and save the number of page and the number of byte where you are at. So the next time an operation needs to be done, the memory can know the location where it has to go.

Port connector:

PWR	DIO	GND	+3V	AIO	IRQ

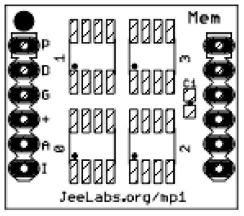
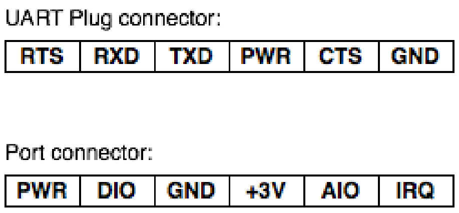


Figure 3 (picture of the memory plug along with the Pinout description.)

- **UART plug:** The UART plug just adds a serial port to the system. It is based on the SC16IS740 chip. It can be interconnected with any devices that needs an UART protocol to communicate up to a baud rate of 230400. It runs using an I2C bus running at 3.3V. So it can be connected in any Jeenode port (Jeelabs, UART plug).


UART Plug connector:

RTS	RXD	TXD	PWR	CTS	GND

Port connector:

PWR	DIO	GND	+3V	AIO	IRQ

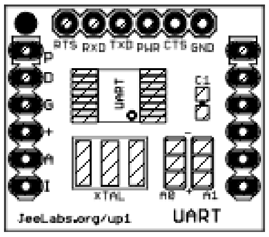
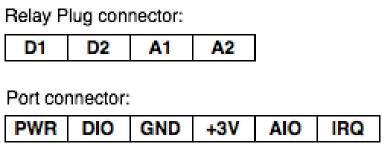


Figure 4 (picture of the UART plug along with the Pinout description)

- **Relay plug:** The relay plug is as the name says two simple relays that can be switched on and off when desired. It connects to the Jeenode by using one of the Jeenode ports (Jeelabs, Relay plug).


Relay Plug connector:

D1	D2	A1	A2

Port connector:

PWR	DIO	GND	+3V	AIO	IRQ

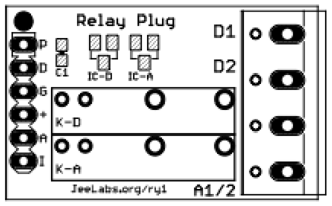
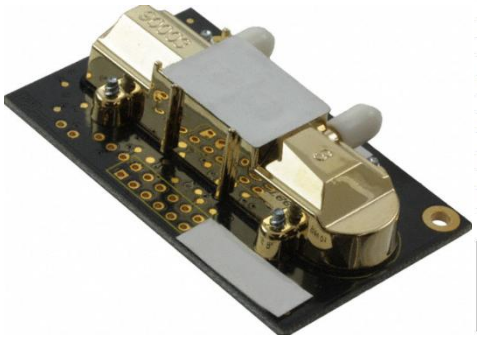


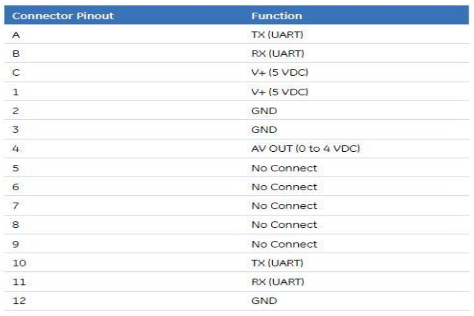
Figure 5 (picture of the relay plug along with the Pinout description)

- **Air pump:** It is a diaphragm pump created by "KNF" which works with a supply voltage of 6V. It possesses two tubing exits and it does not allow any flow of air from the outside to enter in the main chamber of the pump. For more information, look at the datasheet that is on the website listed in the "Work Cited" part of the paper. (KNF)

Figure 6 (picture of the air pump)

- **CO_2 sensor:** For this project, the sensor that is being used is the GE Telaire T6615F which is a dual channel sensor. The sensor works by shining a beam of infrared light, one of the channels measures the reference light that was shined while the other one measures the amount of infrared light that gets back (GE). Then, using Beer-Lambert Law (Wikipedia, Beer_Lambert Law) the concentration of CO_2 can be known. The Telaire 6615F comes with an tight air housing, with two little tubing exits, which does not allow any air to diffuse into the sensor, except by the air that is coming through the tubing. It communicates using UART protocol (Wikipedia, Universal asynchronous receiver/transmitter) with a supply voltage (Vin) of 5V.





Connector Pinout	Function
A	TX (UART)
B	RX (UART)
C	V+ (5 VDC)
1	V+ (5 VDC)
2	GND
3	GND
4	AV OUT (0 to 4 VDC)
5	No Connect
6	No Connect
7	No Connect
8	No Connect
9	No Connect
10	TX (UART)
11	RX (UART)
12	GND

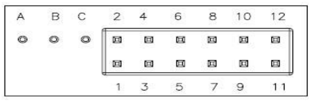


Figure 7 (picture of the Telaire T6615F along with the Pinout description)

- **Temperature and humidity Sensor (Sensirion SHT21):** It is a sensor developed by Sensirion, as the name says it measures the temperature and the humidity on the air. It works using I2C protocol with a SDA and SCL to do the communications with the microcontroller. It has a supply voltage (Vin) of 5V. (Sensirion)

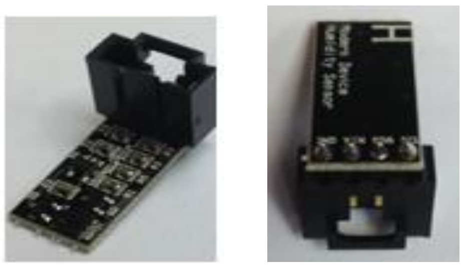


Figure 8 (top-down view of SHT21 sensor)

- **Nafion Tubing:** In simpler terms Nafion is a Teflon backbone with occasional side chains of another fluorocarbon. The side chain terminates in a sulfonic acid (-SO_3H). It works as a water absorbing trap. It can absorb water in its liquid and vapor phase. Each sulfonic acid group can absorb up to 13 molecules of water, which are transported through the membranes of the tubing to the outside of the tubing. (Pure)

Figure 9 (Picture of Nafion tubing)

- **Teflon AF tubing:** The Teflon AF tubing is a type of tubing made out of Teflon AF-2400. This type of material is permeable to many materials, specially to CO_2 (Dasgupta). It is made out of small membranes which allows the partial pressure of CO_2 to equilibrate; therefore, it could act as an equilibrator in an aquatic medium. For this project a 1.6mm OD and 1.0 mm ID was used (Biogeneral).

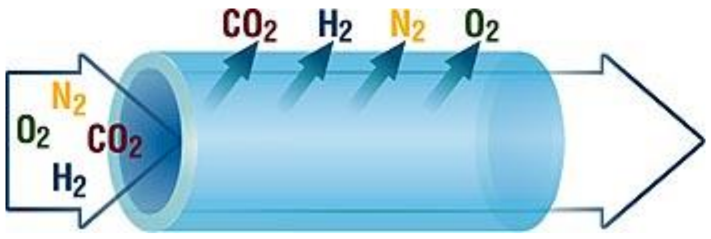


Figure 10 (Teflon AF tubing)

Teflon® AF Gas Permeability, cB *		
Gas	Teflon® AF 2400	PTFE
CO2	280,000	1200
O_2	99,000	420
H_2	220,000	980
N_2	49,000	140
* cB = centiBarrer = 10^{-8} x (cm^2 - cm) / (cm - Hg - sec - cm^2)		

Figure 11 (Permeability of Teflon AF 2400 tubing to different materials)

EXPERIMENTS

- **Determining the equilibration rate of the Teflon AF**

The whole Idea of the project is to create a sensor that can measure CO_2 underwater. In order to do that, it has to be tested if the Teflon AF tubing equilibrates underwater. A series of experiments were done with different concentrations of CO_2 ; first in air and finally underwater, to see if the tubing truly equilibrates, and if it does, find out the rate of equilibration.

The experiment setup was a glass flask with three tubing ports and one plastic cap that can seal completely the inside of the flask. The Teflon AF equilibration tube is placed in the flask and connected to the sensor, and the air pump through two of the ports, the gas supply goes into the last port. The idea was to create a closed loop between the pump, the sensor and the testing environment. The supply was on for a certain amount of time. When turned off the flask was closed so no air could go in or out. Air circulated through the air pump, next to the sensor and then to the flask again. The pump was left running the whole time through the experiment.

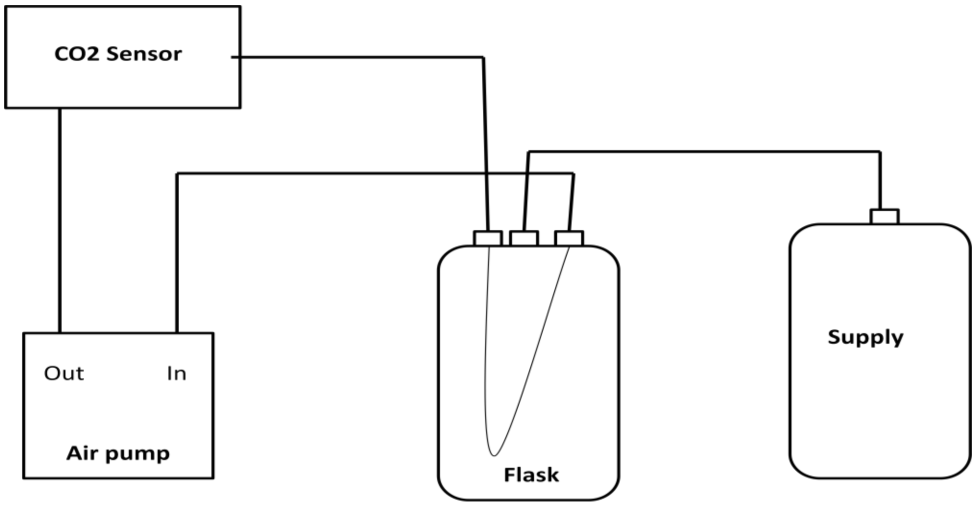


Figure 12(Experiment setup)

- **Equilibration time on air**

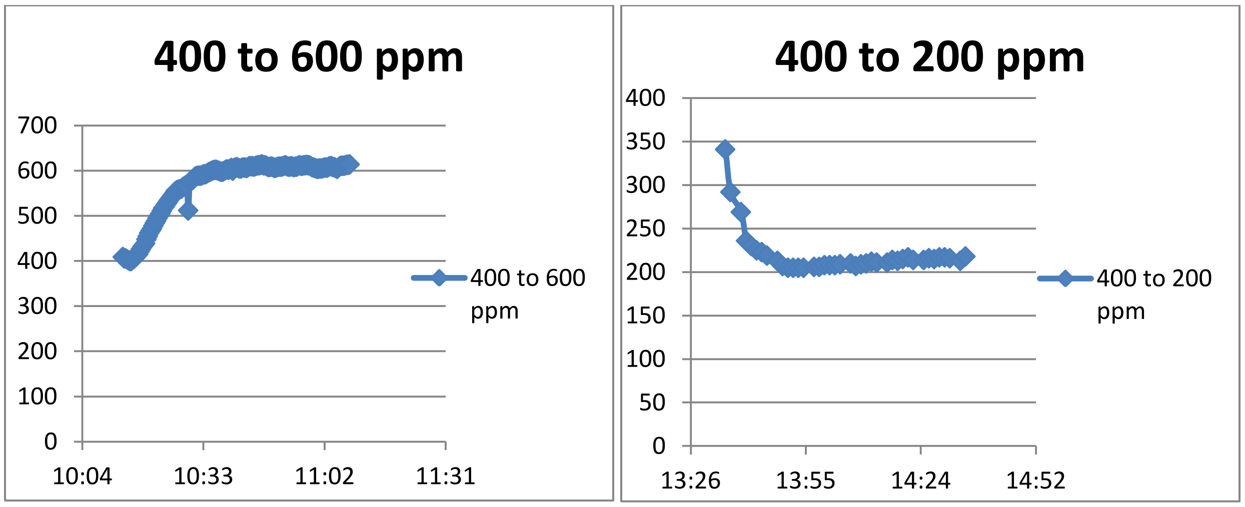
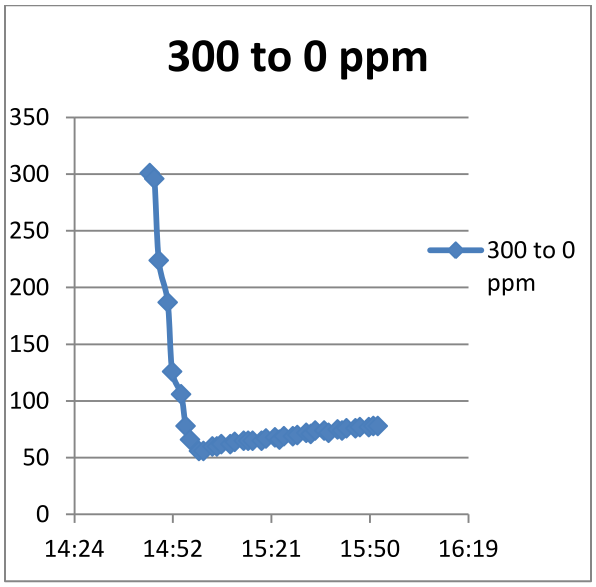
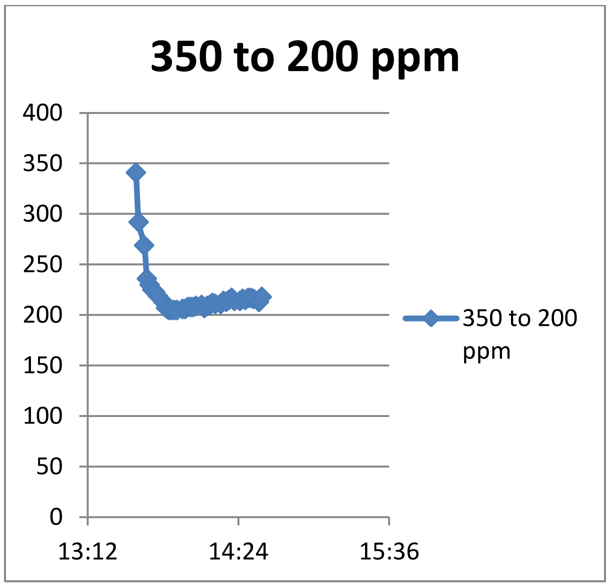


FIgure 13 (Equilibration in air of the Teflon AF tubing, first graph time constant 9.81 min, second graph time constant is 4.3 min)

- **Equilibration time on water**





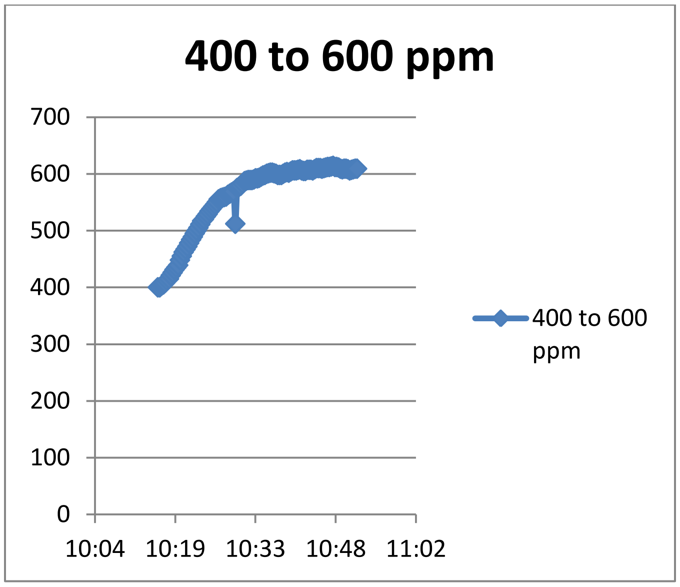


Figure 14(Equilibration in Water of Teflon AF, first graph has a time constant of 4.73 min, Second graph has a time constant of 3.73min, Third graph has a time constant of 8.17 ppm/min)

- **Power consumption:** In order to measure how much time the circuit can run with batteries, first the mAh consumption of the circuit has to be found. A power meter was used in order to look at the amount of power and current that the circuit consumes at each state. The circuit setup used for this experiment is as follows.

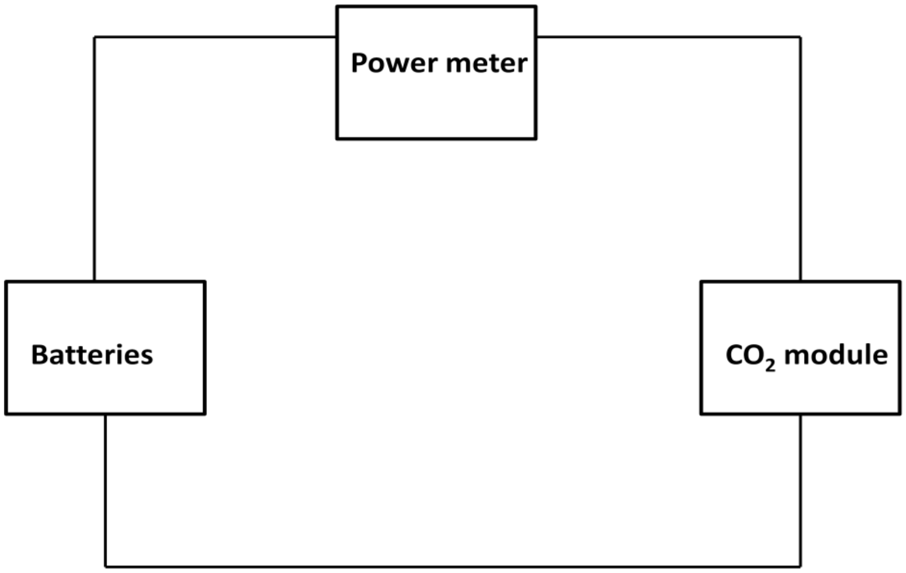


Figure 15 (Circuit to connect the power meter to the sensing module)

After measuring all the devices, the results were as follows:

- Pump consumption: 111mA
- CO_2 Sensor while awake (lights on): 0.145 mA
- Microcontroller +SHT21+ 7805 Voltage regulator+ CO_2 Sensor (sleep mode)=38.3 mA

Knowing how much time each of the stages consume, it would be possible to calculate the total consumption of the sensing module for each possible time scenario. Nevertheless, the total power consumption calculated for this experiment was done for a time cycle of an hour. Where the pump will be turning on for 5 seconds every 3 minutes, the sensor will be in sleep mode for 50 minutes, and finally the sensor will wake up to warm up for 10 minutes and then do a reading.

In order to know the power consumption in mAh of each state knowing the amount of seconds the state will be effective, all what is needed to do is multiply by 1 hour, multiply by the number of seconds the stage will be on, and finally divide by 3600 s.

- Pump consumption: $111mA * 1h * \frac{100s}{3600s} = 3.08mAh$
- Sensor while awake (lights on): $145mA * 1h * \frac{150s}{3600s} = 13050mAs = 6.04mAh$
- Microcontroller +SHT21+ 7805 Voltage regulator+ Sensor (sleep mode):$38.3mA * 1h * \frac{3600s}{3600s} = 38.3mAh$

Total power consumption= 47.42mA.h

If the module is operated with 6 D cell batteries which have an average energy of 17000 mA.h. It would be possible to run it for 14.5 days . Nevertheless, it is recommended to replace the batteries every 13.5 days in order to be sure that the batteries have not started to discharge, therefore causing the circuit to not run properly.

- **Pricing:**

Jeenode	$27
RTC plug	$15
Relay plug	$14
UART plug	$11
Memory plug	$9.50
Telaire 6615	$115
SHT21	$40
Air pump	$290
7805 voltage regulator	$2

Switch	\$4
Teflon AF	\$300
Nafion tubing	\$90
PVC endcap	\$1
PVC tubing	\$1
PVC pipe locker	\$3
Cooper nickel tube	\$5
Total	**\$927**

RESULTS

- **Test tank:** The module was put in a big tank of around 10 meters of deepness. After 12 hours in the test tank, this are the results that were collected

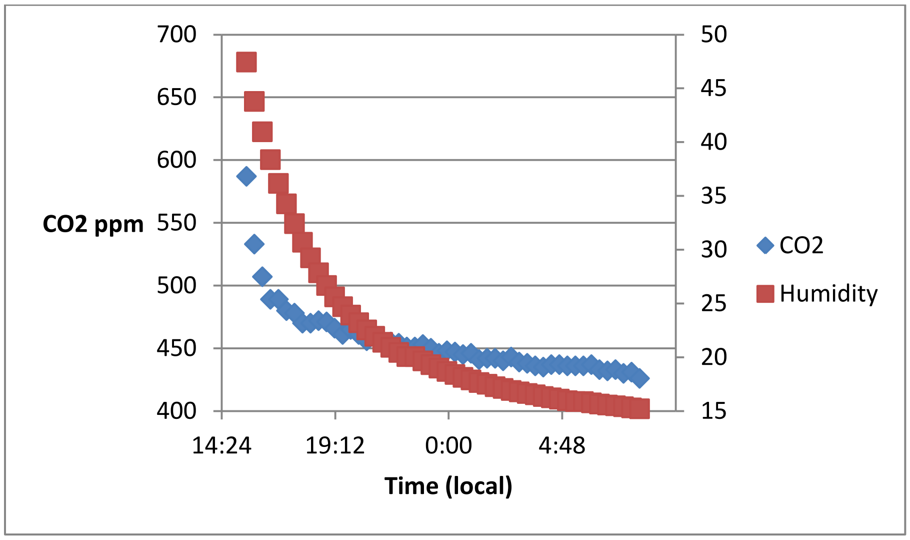


Figure 16 (Results from the test tank)

As visible, the CO_2 seems to be correlated with the humidity; therefore, a comparison between CO_2 and humidity was done. After comparing them, the plot looks as follows.

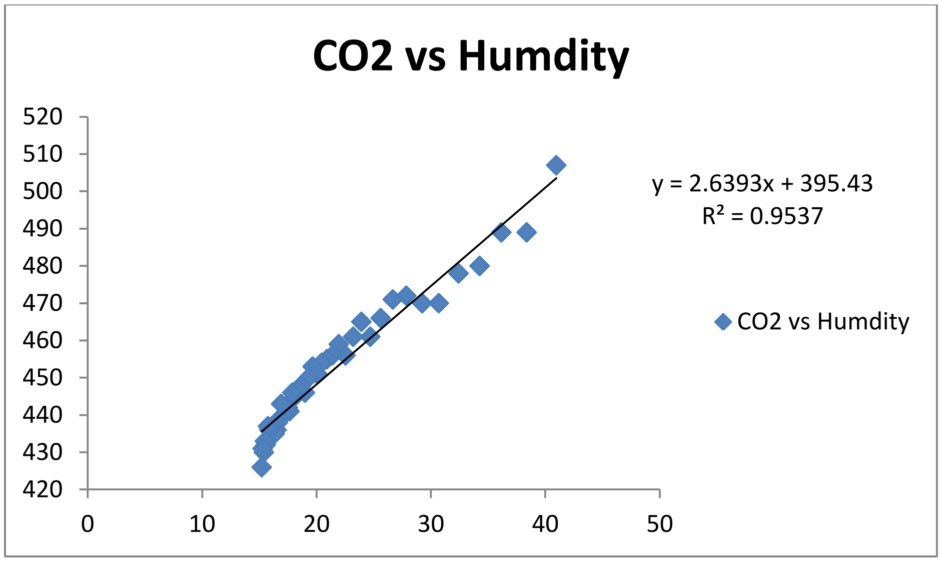


Figure 17 (CO_2)

It seems these two parameters have a linear correlation; therefore a humidity correction needs to be done. The CO_2 values will be readjusted to zero humidity using the slope of 2.6 ppm/%humidity (Figure17).

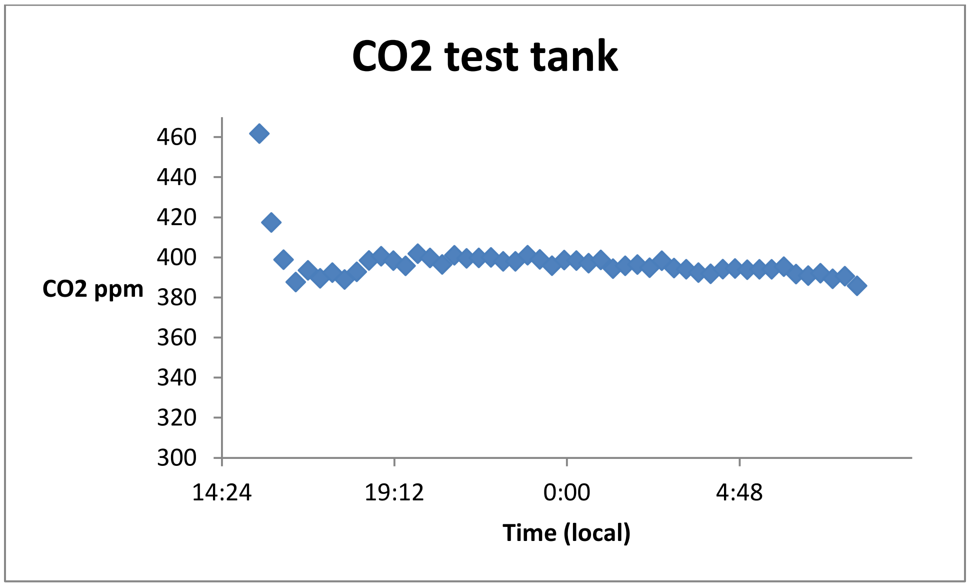
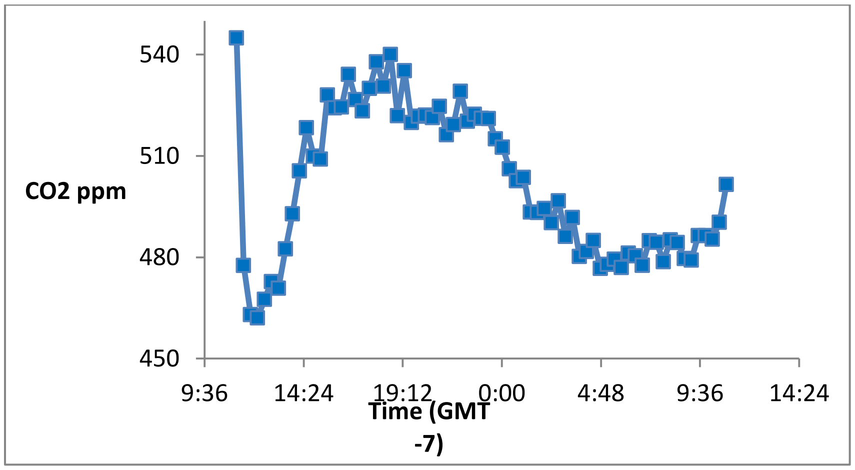
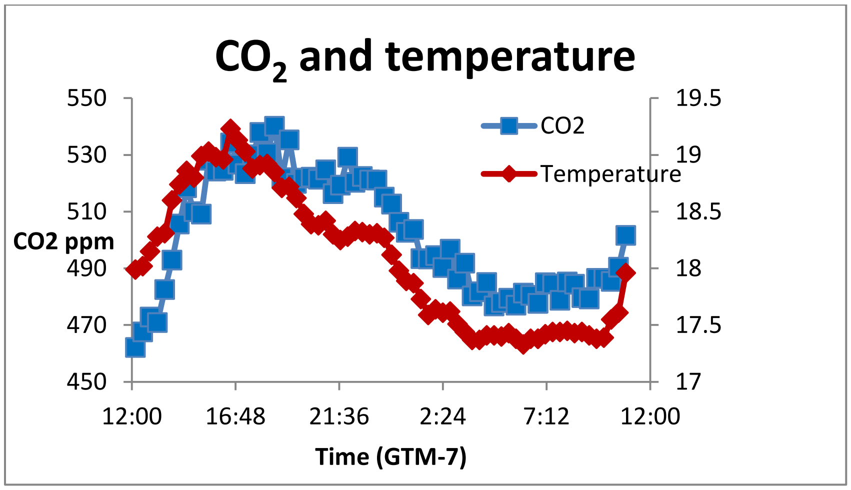


Figure 18 (Results from the test tank (humidity corrected))

- **Harbor:** The module was put in the Moss Landing, CA harbor for 24 hours. The data collected was first humidity corrected as it was done on the test tank results, and afterwards, it was analyzed in the three following ways.





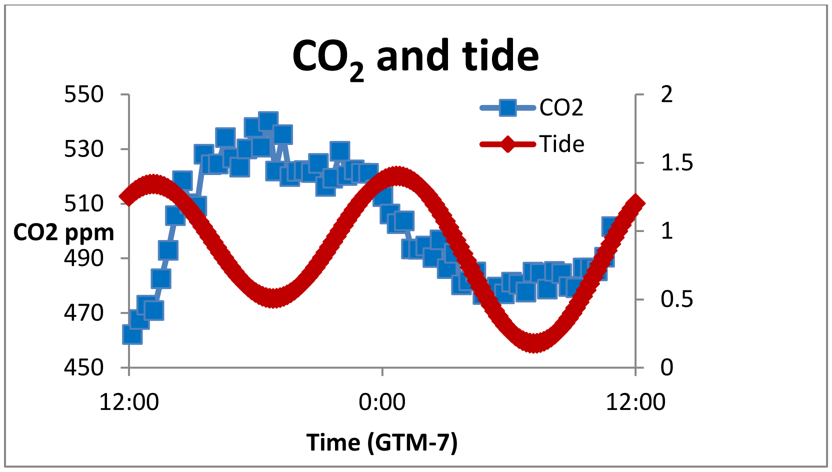


Figure 19 (Results from the Moss Landing harbor)

TUTORIAL

In this tutorial, it will be demonstrated how to build your own CO_2 module and how to make it work underwater. All the steps are shown, from building the module, how to calibrate the sensor if it goes out of calibration, how to build the water tight housing, and how to hookup all the tubing necessary to measure CO_2 .

- **Assembling the Jeenode**: As described before, the Jeenode has different ports for connecting different devices. Solder the following devices in the following ports: RTC plug in Port 1, Relay plug in Port 2, Uart plug in Port 3, and Memory plug in Port 4 (remember to connect the right pins with the right holes of the devices, just make sure the names of the holes are the same as the names on the Jeenode). Remember to connect the communication antenna (simply a wire) on the hole that says "A" besides the RF12 board

- **Assembling the circuitry together:** The final circuit is divided into two sub circuits.

▪ **Power lines**: first, make sure that all the devices in the circuit are getting the voltage that they need to operate. A 7805 voltage regulator will be used, in order to provide the 5 volts that the Jeenode, the Telaire CO_2 sensor and the SHT21 need to properly function. In addition, add an electrolytic capacitor of 100 uF from the power of the Jeenode to ground, this will help the circuit to properly function when the batteries get a little low. The power line circuit looks as follows. **Before assembling this circuit please read the section about "battery pack" where more details will be explained about how to use the 7805 5V regulator**

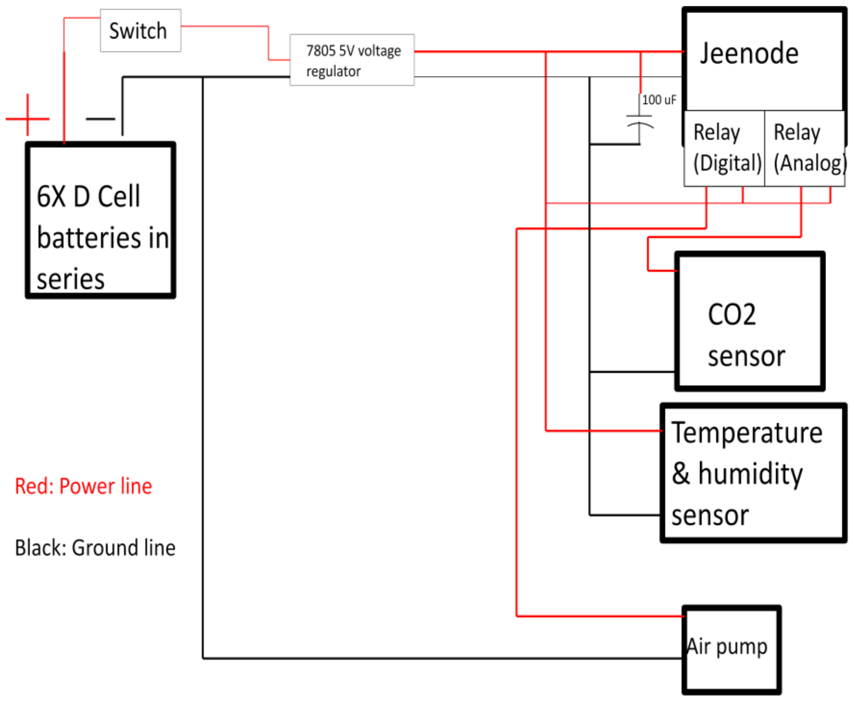


Figure 20 (Block diagram of the circuit for the power lines)

- **Communication lines**: The communication lines are the lines that are made so that the microcontroller (the Jeenode) can communicate with the Telaire sensor and the SHT21. The Telaire sensor uses UART protocol in order to communicate with the microcontroller; therefore, it has to have an TX and RX connection with the microcontroller. The SHT21 uses I2C protocol in order to communicate with the microcontroller; therefore, it has to have an SDA and SCL connection with the microcontroller. The communication line is as follows.

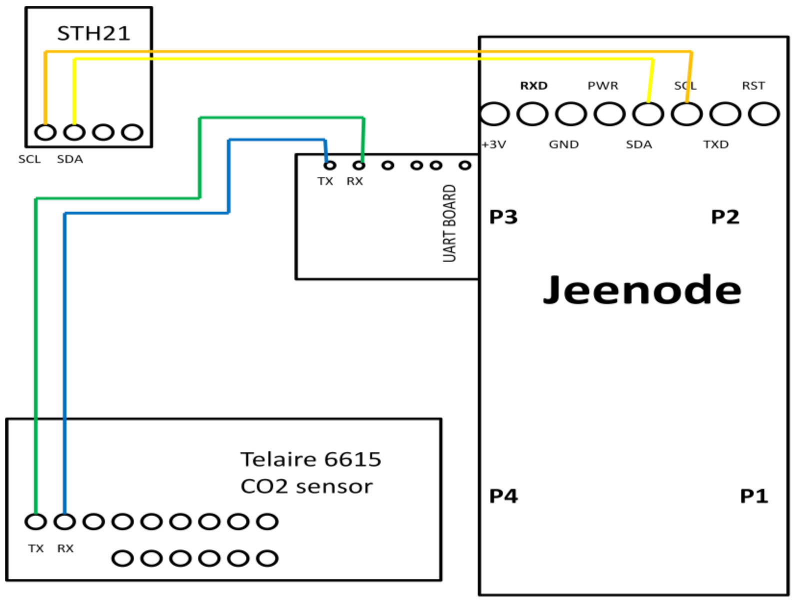


Figure 21(Block diagram of the circuit for the communication lines)

When everything is assembled, the end result should be something looking as follows:

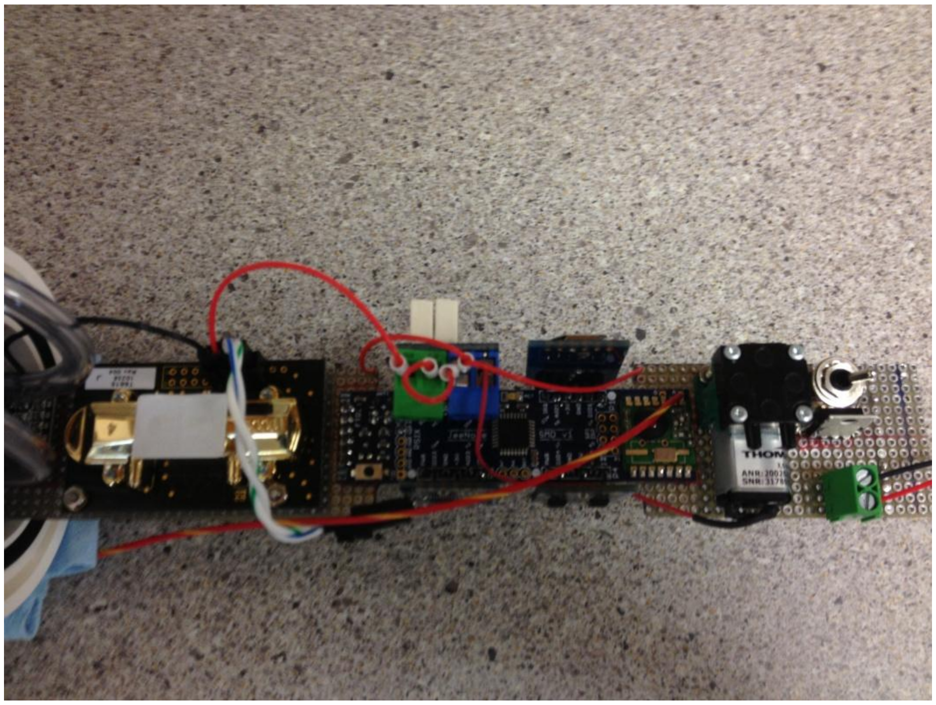


Figure 22 (Actual CO_2 sensor module)

- **Battery pack:** Since the module is using a 7805 5 Volts regulator, a voltage greater than 7.5V and less than 20V has to be used in order to have a good regulation. The supply voltage will be 9V to allow the circuit to keep working even when the batteries start getting discharged. This will give us a 1.5V offset of discharge until the circuit stops working. The 7805 requires special circuitry in order to work, it just needs two capacitors. There are different circuits that can be used to do a 5V regulation with the 7805 regulator; nevertheless, this was the circuit used for this project:

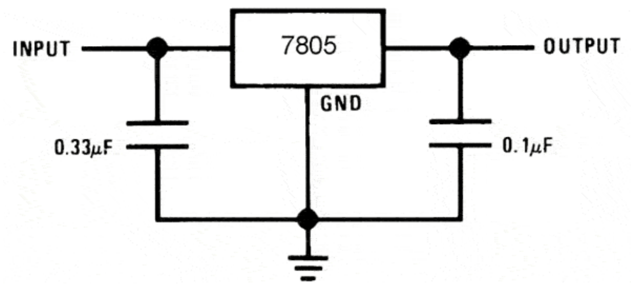


Figure 23(7805 5V regulation circuit)

- **Housing:** The CO_2 sensing module and battery pack are designed to go inside A PVC tube. The sensing module is attached to a PVC endcap through screw holes so it could be stable, and it does not bend too much (If bended too much , the sensing module board could break). There are also two barb fittings that are used as entrance and exit of the tubing system for the CO_2 module.

  The outside of the endcap has a copper nickel tube in which Teflon AF tubing can go inside. The cooper nickel tube is going to go attached with the endcap with two swage locks . The last component on the outside are two compression fittings through which the Teflon AF is going to go in and out of the system.

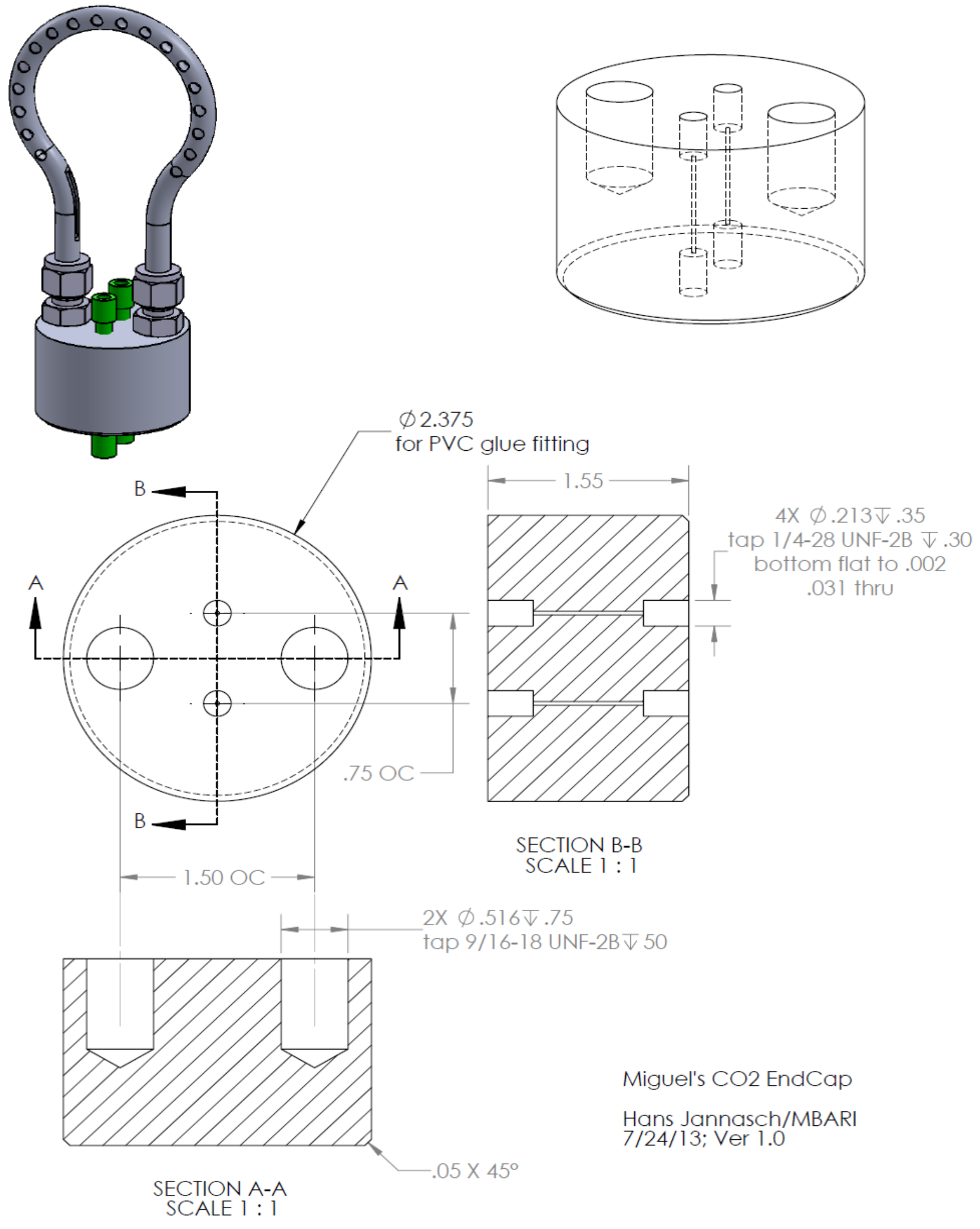


Figure 24 (Endcap design)

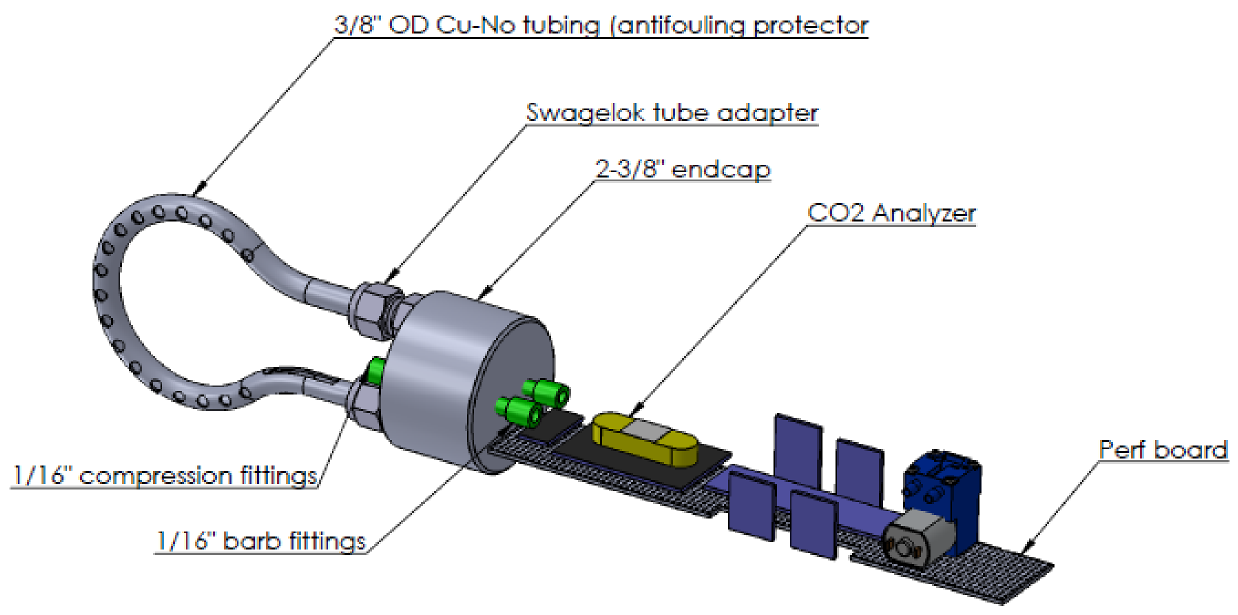


Figure 25 (Assembled endcap)

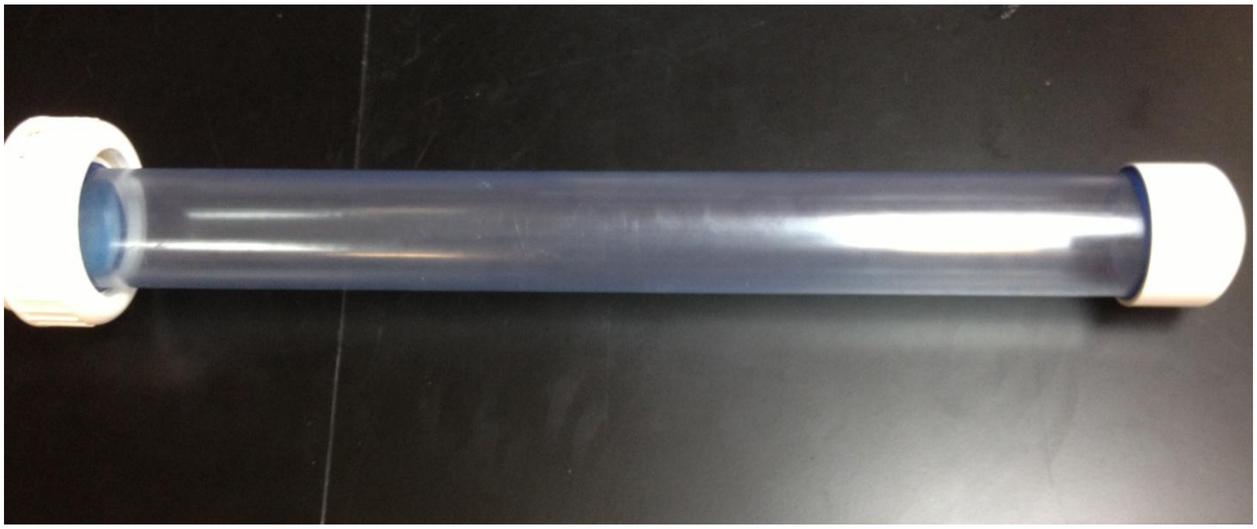
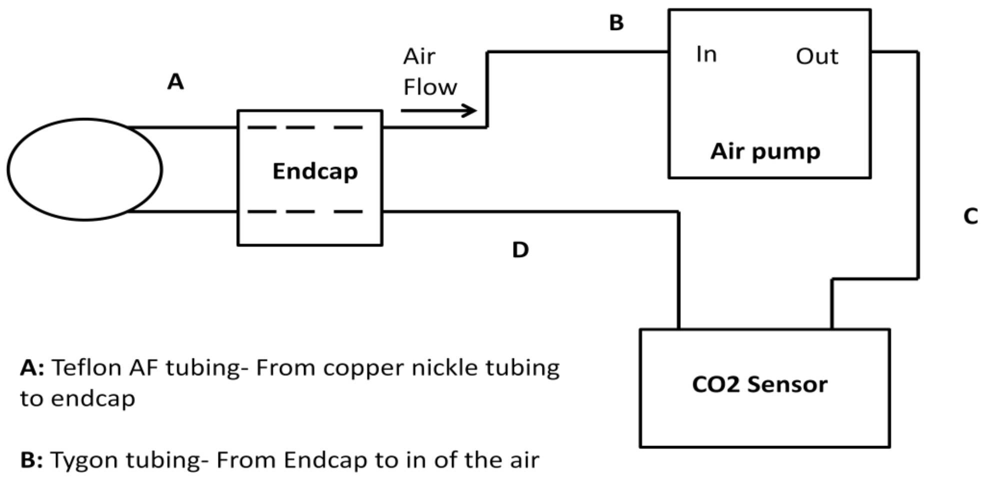


Figure 26(PVC tubing and PVC pipe union)

- **Tubing layout:** From Figure 21, it is visible that the Teflon AF tubing is going to go inside of the Cooper nickel tubing, get out of it and go into the compression fittings. It is recommendable that at least two loops are made inside of the cooper nickel tubing, the more surface area there is of tubing, then the faster the equilibration process. For the inside part of the endcap, is no longer necessary to use more Teflon AF since no equilibration is going to take place inside the housing. After the barb Fittings, it is recommendable to use Tygon tubing. Do not forget about using the Nafion tubing before going into the sensor, as water vapor could interfere with CO_2 measurements. Silica gel dryers packs must be placed inside the pressure housing to reduce humidity on the outside of the Nafion to dry the gas inside.

9



A: Teflon AF tubing- From copper nickle tubing to endcap

B: Tygon tubing- From Endcap to in of the air pump

C: Nafion Tubing- From out of the air pump to CO2 sensor

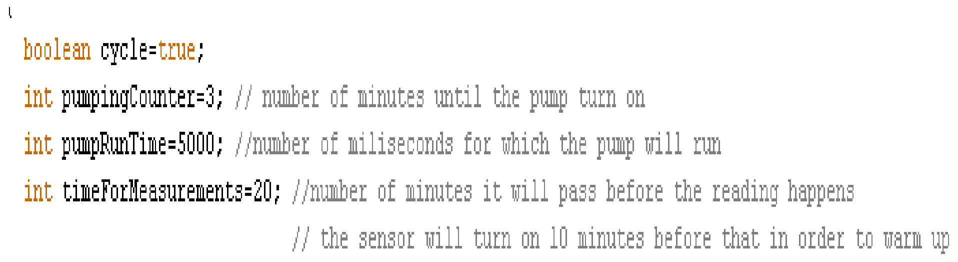
D: Tygon tubing- From the CO2 sensor to the Endcap

Figure 27 (Tubing layout)

- **Code:** The code is made so it can be as user friendly as possible. It is subdivided into three main codes, one for the transmitter, one for the CO_2 sensing module, and one for the calibration and sensor functionality. The complete code of the transmitter and the CO_2

sensing module can be found in appendix A and B respectively. For looking at the code of the sensor calibration and functionality, please visit appendix C.

- **CO_2 sensing module:** This code is composed of mainly three functions: read_Save(), load_Memory (char buf) , erase_Memory, and rf12_Interrupt(). The read_Save() part of the code, as the name says, it just reads data and saves it into the memory. It works using cycles predefined by the user, where the user can specified the pumping rate and the amount of minutes between each measurement (notice that before a measurement can be done, the sensor has to warm up at least 10 minutes, that timing system is already included in the code). If the user wants to control the timing of the pump and sensor the following lines have to be changed

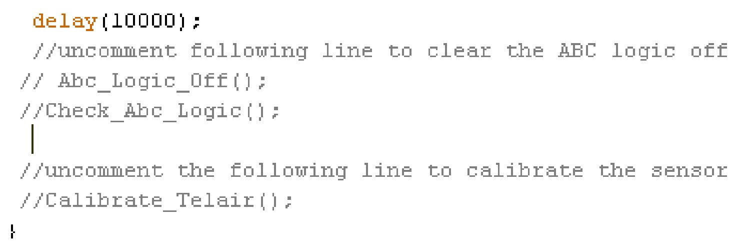


```
boolean cycle=true;
int pumpingCounter=3; // number of minutes until the pump turn on
int pumpRunTime=5000; //number of miliseconds for which the pump will run
int timeForMeasurements=20; //number of minutes it will pass before the reading happens
                             // the sensor will turn on 10 minutes before that in order to warm up
```

Figure 28 (Code example of cycle timing)

- **Transmitter:** The transmitter could be just another Jeenode or a Jeelink (same as a Jeenode with the only difference that is more compact and no plugs can be attached to it). The transmitter is set up so when the user enters a "1" through the serial port, the Sensing module will start to send the data loaded into memory to the transmitter, and then resume normal operation. If the user enters a "2", the sensing module will erase the data inside the memory.

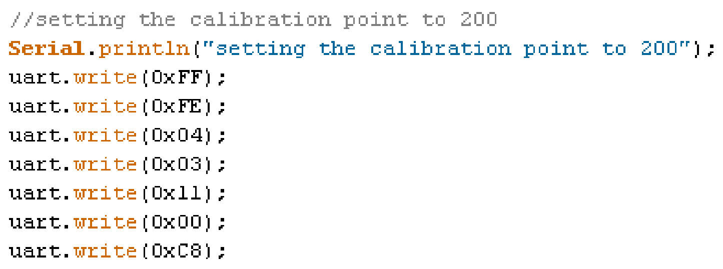
- **Sensor calibration and functionality:** The sensor calibration and functionality code is just used when the sensor wants to be calibrated, turn the ABC logic off, or test the functionality. In order to tell the sensor to do either calibration or turn off the ABC logic, some lines have to be commented or uncommented; this lines are shown in Figure 25. If the sensor has to be calibrated, the gas concentration has to be introduced. In the code the setup is to do it at 200 ppm. nevertheless, if the concentration has to be changed, the last two lines of the command shown in Figure 26 have to be changed into the number that was to be input in hexadecimal notation.



```
 delay(10000);
 //uncomment following line to clear the ABC logic off
// Abc_Logic_Off();
//Check_Abc_Logic();

//uncomment the following line to calibrate the sensor
//Calibrate_Telair();
}
```

Figure 29



```
//setting the calibration point to 200
Serial.println("setting the calibration point to 200");
uart.write(0xFF);
uart.write(0xFE);
uart.write(0x04);
uart.write(0x03);
uart.write(0x11);
uart.write(0x00);
uart.write(0xC8);
```

Figure 30

Example: The setup for the calibration is 200 ppm which is represented as 00C8 in hexadecimal; therefore the last two lines of Figure 26 are

uart.wrtie(0x00);

```
uart.write(0xC8);
```

If the number wants to be changed two 400, the hexadecimal representation would be: 0190; therefore, the last two lines of code in Figure 26 would be as follows:

```
uart.wrtie(0x01);
uart.write(0x90);
```

Note: After the calibration process is done, weird data will start appearing on the serial monitor, please wait one minute after the calibration process, and simply turn off the sensor, next time it is turned on, it should be calibrated.

- **User interface:** A LabView user interface has been created in order to send the radio transmission commands as easy as possible. The LabView interface comes with real time charts that will allow the visualization of data in real time. The data is also imported to an Excel file if wanted. If the user interface wants to be used, all what it has to be done is open the executable file named CO2 _sensing_module.exe. Nevertheless, sending the serial commands through the Arduino environment is also a good solution.

Conclusion

In this paper it was proved that a reliable, low power, low cost underwater CO_2 sensing module could be built in an easy way. All the parts can be easily obtained online at not with an elevated price, if compared to the professional CO_2 sensing modules that are already on the sea. In order to assemble the module some technical skills are needed. Nevertheless, the tutorial presented on this paper is made to be as explicative as possible.

The results from this project showed that Teflon AF is an excellent CO_2 equilibrator on air and underwater. How fast equilibrates is going to be proportional to the flow of air that has through it, how long the tubing is, and how big the diameter is. For underwater research, it has to be evaluated and tested the amount of pressure it can withstand , in order to know how deep it could be taken. In addition a better algorithm and hardware could be developed in order to make the sensing module as low power consuming as possible.

To conclude, it has to be said that this project serves as the base for a better platform that will allow better readings of CO_2 with a longer battery life period. More research has to be done in order to find a better sensor that can detect CO_2 with more precision and less consumption of power. However, the project presented on this paper fulfills the objective of being easy to build so it can be used for education and little research projects.

REFERENCES

ATABEK, SINAN. *CO2 sensors*. n.d. document. 1 August 20123.

Biogeneral. *Teflon® AF Tubing and Membrane*. n.d. BIOGENERAL. Web page. 2 August 13. <http://www.biogeneral.com/teflon.html (picture and table of Teflon AF tubing)>.

Damage to Marine Ecosystems as CO2 Emissions Rise. n.d. Save the seas Foundation. web page. 1 August 2013. <http://saveourseas.com/articles/damage_to_marine_ecosystems_as_co2_emissions_rise>.

Dasgupta, Purnendu. *High-Sensitivity GAs Sensor Based on Gas-Permeable Liquid Core Waveguides And Long-Path Absorbance Detection*. Lubbock, 1998. Document.

GE. *Telaire 6615*. n.d. GE, Measurment & Control. 1 August 2013. <http://www.ge-mcs.com/download/co2-flow/920-474D-LR.pdf>.

Jeelabs. *Jeenode SMD*. n.d. Jeelbas. Web page. 1 August 2013. <http://jeelabs.net/projects/hardware/wiki/JeeNode_SMD>.

—. *Memory plug*. n.d. Web page. 1 August 2013. <http://jeelabs.net/projects/hardware/wiki/Memory_Plug>.

—. *Relay plug*. n.d. Web page. 1 August 2013. <http://jeelabs.net/projects/hardware/wiki/Relay_Plug>.

—. *RTC plug*. n.d. Web page. 1 August 2013. <http://jeelabs.net/projects/hardware/wiki/RTC_Plug>.

—. *UART plug*. n.d. Web page. 1 August 2013. <http://jeelabs.net/projects/hardware/wiki/UART_Plug>.

KNF. *Micro Diaphragm Gas Sampling Pump*. 2013. THOMAS. Web Page. 5 August 2013. <http://www.gd-thomas.com/product.aspx?id=13052&tp=p>.

Mitchell, Alan. *Alan Mitchell Building Energy Use Monitoring and Analysis*. n.d. 1 August 2013. <http://alanbmitchell.wordpress.com/2011/10/02/operate-arduino-for-year-from-batteries/>.

NASA. *Effects of changing the Carbon Cycle*. n.d. Web. 1 August 2013. <http://earthobservatory.nasa.gov/Features/CarbonCycle/page5.php>.

Ning, Liu, et al. *Development of a CO2 chemical sensor and sensor strategy for CO2 monitoring in carbon sequestration*. n.d. Web document. 1 August 2013. <http://baervan.nmt.edu/research_groups/carbon_sequestration_membrane_technology/pages/Publications/B-1.pdf>.

Pure, Perma. *Perma Pure FAQs*. n.d. Perma Pure LLC. Web page. 2 August 2013. <http://www.permapure.com/support/faqs/.>.

Sensirion. *SHT21 - Digital Humidity Sensor (RH&T)*. n.d. Sensirion. Web page. 2 August 2013. <http://www.sensirion.com/en/products/humidity-temperature/humidity-sensor-sht21/>.

Wikipedia. *Beer_Lambert Law*. n.d. Web document. 5 8 2013. <http://en.wikipedia.org/wiki/Beer%E2%80%93Lambert_law>.

—. *I2C*. n.d. Web document. 05 August 2013. <http://en.wikipedia.org/wiki/I%C2%B2C>.

—. *Universal asynchronous receiver/transmitter*. n.d. Web document. 5 8 2013. <http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter>.

APPENDIX

- **Appendix A**

```
#include <JeeLib.h>

// set up the timer for the radio communications
MilliTimer sendTimer;

//global variables
byte needToSend;

void setup()
{
  Serial.begin(9600);
  rf12_initialize(1,RF12_915MHZ,1);
}

void loop()
{
  send_Command();
}


/*****************************************************************
 * Function name: send_Command()
 * type: void
 * input parameters: NULL
 * output parameter: NULL
 * Description: Sends a command, wait for response of the node, and then
 * make an action according to the command receicved. If a 1
 * is sent, loading of the memory will take place. If a 2 is
 * sent, deletion of the memory will take place.
 * Creators: MBARI
 *****************************************************************/
void send_Command()
{
  boolean out=true;
  boolean receiveCommand=true;
  uint8_t buf[1];
  char reading[1];
  uint8_t data[12];

  while(receiveCommand)
  {
    if(Serial.available())
    {
      reading[0]=Serial.read();
      if(reading[0]=='1'||reading[0]=='2')
      {
        reading[0]-=48;
        receiveCommand=false;
      }
    }
  }

  while(out)
  {
    if (rf12_recvDone() && rf12_crc == 0) // received good data if true
    {
      buf[0]=rf12_data[0];
      if(buf[0]==1)
      {
        while(1)
        {
          if (rf12_recvDone() && rf12_crc == 0) // received good data if
true
          {
            for (byte i = 0; i < rf12_len; ++i)
            {
              data[i]=rf12_data[i];
            }
            if(data[0]==50)
            {
              Serial.println("memory empty");
            }
            else
            {
              transform(data);
            }
          }
        }
      }
      else if(buf[0]==2)
      {
        Serial.println("memory erased");
      }
      out=false;
    }
    else
    {
      if (sendTimer.poll(10))
      {
        needToSend = 1;
      }
      if (needToSend && rf12_canSend())
      {
        rf12_sendStart(2,&reading,1);
        needToSend = 0;
      }
    }
  }
}

/*******************************************************************
 * Function name: transform(char [])
 * type: void
 * input parameters: uint8_t a[]: array of n bytes which will be loaded
 * output parameter: NULL
 * Description: Transform an arry of 1 byte integers into something that
looks
 * like mm//dd//yy h:m:s CO2 ppm= xxx temperature= xx.xx
 * humidity= xx.x
 * Creators: MBARI
 ****************************************************************/
void transform( uint8_t a[])
{
  int buffer[11];
  char string[80];
  for(int i=0;i<6;i++) //get the time form the clock
  {
    buffer[i]= (int)a[i];
  }
  buffer[6]=((a[6] &0xff)<<8) | (a[7] &0xff); //merge the two bytes of co2

  for(int i=8;i<12;i++) // get the values for temperature and humidity
  {
    buffer[i-1]=int(a[i]);
  }

  sprintf(string,"%02d/%02d/%02d %02d:%02d:%02d  CO2 ppm= %02d
temperature= %02d.%02d humidity= %02d.%02d",

buffer[0],buffer[1],buffer[2],buffer[3],buffer[4],buffer[5],buffer[6],buff
er[7],buffer[8],buffer[9],buffer[10]);
  Serial.print(string);
  Serial.println();
  delay(1);
}
```

- **Appendix B**

```
#include <Wire.h>
#include <JeeLib.h>
#include <LibHumidity.h>

// sets up the clock port and the clock module
PortI2C myport (1 /*, PortI2C::KHZ400 */);
DeviceI2C rtc (myport, 0x68);

//sets up the port of the relay
Port relays (2);

//sets up the port of the uart and the uart module
PortI2C i2cBusUart (3);
UartPlug uart (i2cBusUart, 0x4D);

//sets up the port of the memory and the memory module
PortI2C i2cBusMem (4);
MemoryPlug mem (i2cBusMem);
MemoryStream stream (mem);

//sets up the Sensirion sensor for temperature and humidity
LibHumidity humidity_Temp = LibHumidity(0);

MilliTimer sendTimer; //setup the time for the radio communications



// must be defined in case we're using the watchdog for low-power waiting
ISR(WDT_vect) {
  Sleepy::watchdogEvent();
}


//global variables
char bufData[12];//buffer for the data is going to go into the memory

void setup()
{
  Serial.begin(9600);
  uart.begin(19200);


  // initialize the radio for this node
  rf12_initialize(2, RF12_915MHZ, 1);

  //initialize the relays
  relays.digiWrite(0);
  relays.mode(OUTPUT);
  relays.digiWrite2(0);
  relays.mode2(OUTPUT);
  //uncomment the next two lines when you wat to use the memory for the
first time
  //this will start the directory to the page 0 adress 0
  //after compiling and running it for 2 seconds comment them again and
compile the code again
  //int buf[2]={0,0};
  //mem.save(0,0,buf,4);

  //uncomment he following line to set up the date and time for the clock
for the first time
  //run the code for the 1 seconds comment the following line again and
compile
  //setDate (13,8,9,10,03,0);

  /*while(1)
  {
    if(Serial.read()=='1')
    {
      char buf[12];
      load_Memory(buf);
    }
  }*/
  delay(10000);
  Serial.end();
}

void loop()
{
  read_Save();
}

/*******************************************************************
 * Function name: read_Save(void)
 * type: void
 * input parameters: void
 * output parameter: NULL
 * Description: function that records a Co2 value from the telaire sensor
 * and stores it into memory
 * Creators: MBARI
 *********************************************************************/
void read_Save()
{
  boolean cycle=true;
  int co2Value=0;
  int pumpingCounter=3; // number of minutes until the pump turn on
  int pumpRunTime=5000; //number of miliseconds for which the pump will
run
  int timeForMeasurements=20;
  boolean idle=true; //number of minutes it will pass before the reading
happens
  // the sensor will turn on 10 minutes before that in order to warm up
  int counter=0;
  byte time[6];
  int i=0;
  //send the Telair to sleep
  relays.digiWrite2(1);//turn on the sensor
  delay(10000);
  Telaire_IdleMode(idle);

  //clearing the memory data array
  for(int j=0;j<12;j++)
  {
    bufData[j]=NULL;
    j++;
  }

  while(cycle)
  {
    //check for communication
    rf12_sleep(-1);
    rf12_recvDone();
    receive_Command();
    rf12_sleep(1);
    rf12_sleep(0);



    //sleep for n number of seconds if the pump does not turn on during
the iteration
    if(pumpingCounter!=3)
    {
      Sleepy::loseSomeTime(pumpRunTime);
    }

    // turn the pump on for n number of seconds
    else if(pumpingCounter==3)
    {
      relays.digiWrite(1);
      Sleepy::loseSomeTime(pumpRunTime);
      relays.digiWrite(0);
      pumpingCounter=0;
    }
    Sleepy::loseSomeTime((word)(60000-pumpRunTime));
    counter++;
    pumpingCounter++;

    if(counter==timeForMeasurements-10)
    {
      Telaire_IdleMode(!idle);
    }

    if(counter==timeForMeasurements)
    {
      Serial.begin(9600);
      delay(20);
      co2Value=read_Co2();
      while((co2Value>2000||co2Value<=0)&&i<=10)
      {
        co2Value=read_Co2();
        if(i==10)
        {
          co2Value=1;
        }
        i++;
        delay(10000);
      }
      float temp=get_Temperature();
      delay(20);
      float humidity=get_Humidity();

      //getting the date and time in the buffer for the memory
      getDate(time);
      bufData[0]=time[1];
      bufData[1]=time[2];
      bufData[2]=time[0];
      bufData[3]=time[3];
      bufData[4]=time[4];
      bufData[5]=time[5];
      delay(20);

      Serial.print(time[1]);
      Serial.print("/");
      Serial.print(time[2]);
      Serial.print("/");
      Serial.print(time[0]);
      Serial.print(" ");
      Serial.print(time[3]);
      Serial.print(":");
      Serial.print(time[4]);
      Serial.print(":");
      Serial.print(time[5]);
      Serial.print(" ");
      Serial.print(co2Value);
      Serial.print(" ");
      Serial.print(temp);
      Serial.print(" ");
      Serial.print(humidity);
      Serial.println();
      saved(bufData);
      delay(10);
      cycle=false;
      counter=0;
      Serial.end();
    }
  }
}



/******************************************************************
 * Function name: load_Memory(char [])
 * type: void
 * input parameters: char a[]: array of n bytes which will be loaded
 * output parameter: NULL
 * Description: Loads the the information that is on memory and transmit
 * it thorugh radio communications to a base station.
 * Creators: MBARI
 ******************************************************************/

void load_Memory(char a[]) //function for loading characters in a fast way
{
  int finish[2];
  int remember[2]={
    1,0    };
  if(mem.isPresent())
  {
    mem.load(0,0,finish,4);
    delay(20);
    if(finish[0]!=0 && finish[1]!=0)
    {
      while((remember[0]!=finish[0])||(remember[1]!=finish[1]))
      {
        mem.load(remember[0],remember[1],a,12);
        delay(50);
        send_Radio(a,12);
        delay(1);
        transform(a);
        if(remember[1]<243)
        {
          remember[1]+=12;
        }
        if(remember[1]>243)
        {
          remember[0]+=1;
          remember[1]=0;
        }
      }
    }
    else
    {
      Serial.println("memory empty");
      uint8_t checkDeletion[1]={50};
      rf12_sendStart(1,checkDeletion,1);
    }
  }
}

/*****************************************************************
 * Function name: erase_Memory()
 * type: void
 * input parameters: void
 * output parameter: NULL
 * Description: Erases the memory.resets the directory making the memory
 * to start overriding previous values every time it saves
 * new data
 * Creators: MBARI
 *******************************************************************/
void erase_Memory()
{
  int erase[2]={0,0};
  if(mem.isPresent())
  {
    mem.save(0,0,erase,4);
    Serial.println("memory erased");
  }
}


/*******************************************************************
 * Function name: receive_Command()
 * type: void
 * input parameters: NIULL
 * output parameter: NULL
 * Description: Checks if there is any radio communication command
 * from the basen station and takes and action according
 * to the command
 * Creators: MBARI
 *******************************************************************/
void receive_Command()
{
  uint8_t buf[1];
  char buffer[12];
  byte InfoReceived=0;
  byte needToSend=0;
  int i=0;
  boolean out=true;

  Serial.begin(9600);
  delay(20);
  while(i<50 && out)//i<2000 && out
  {
    i++;

    if (rf12_recvDone() && rf12_crc == 0) // received good data if true
    {
      buf[0]=rf12_data[0];
      InfoReceived=1;
      needToSend=1;
    }

    if(needToSend && rf12_canSend() && InfoReceived)
    {
      rf12_sendStart(1, buf,1);
      InfoReceived=0;
      needToSend = 0;
      out=false;
      if(buf[0]==1)
      {
        load_Memory(buffer);
      }
      if(buf[0]==2)
      {
        erase_Memory();
      }
    }
  }
  Serial.end();
  delay(20);
}

/*****************************************************************
 * Function name: read_Co2(void)
 * type: int
 * input parameters: void
 * output parameter: int
 * Description: tells tehe sensor to start doing measurments and store
 * the two bytes in a varible called co2value
 * Creators: MBARI
 ******************************************************************/
int read_Co2()
{
  byte buf[56];
  byte i=0;
  int co2Value;

  uart.write(0xFF);
  uart.write(0xFE);
  uart.write(0x02);
  uart.write(0x02);
  uart.write(0x03);
  delay(20);

  // put he two bit answer from the sensor into a buffer
  while(uart.available())
  {
    buf[i]=uart.read();
    i++;
  }

  //save the two bits of Co2 into memory buffer
  bufData[6]=buf[3];
  bufData[7]=buf[4];

  // do a bitwise operation to acomodate the two bits
  // and make them an integer
  co2Value=0;
  co2Value|=buf[3] & 0xFF;
  co2Value=co2Value<<8;
  co2Value|=buf[4] & 0xFF;
  return co2Value;
}

/*****************************************************************
```

```
 * Function name: get_Temp(void)
 * type: float
 * input parameters: void
 * output parameter: float
 * Description: tells the Sensirion sensor to start doing measurements
 * of temperature, after that it divides the integer and
 * float part of the number in two bytes so they can be
 * stored into memory. The function uses the library
 * LibHumidity from www.moderndevices.com in order to control
 * the sensor
 * Creators: MBARI
 *******************************************************************/
float get_Temperature()
{
  float temp;
  byte tempInt;
  byte tempFloat;

  /* get the measurement of temperature and divides them into an
   integer part and a float part*/
  temp=humidity_Temp.GetTemperatureC();
  tempInt=floor(temp);
  tempFloat=(temp*100)-(tempInt*100);

  //get the date into memroy
  bufData[8]=tempInt;
  bufData[9]=tempFloat;

  return temp;
}

/******************************************************************
 * Function name: get_Humidity(void)
 * type: float
 * input parameters: void
 * output parameter: float
 * Description: tells the Sensirion sensor to start doing measurements
 * of humidity, after that it divides the integer and
 * float part of the number in two bytes so they can be
 * stored into memory. The function uses the library
 * LibHumidity from www.moderndevices.com in order to control
 * the sensor
 * Creators: MBARI
 *******************************************************************/
float get_Humidity()
{
  float humidity;
  byte humidityInt;
  byte humidityFloat;

  /* get the measurement of humidity and divides them into an
   integer part and a float part*/
  humidity=humidity_Temp.GetHumidity();
  humidityInt=floor(humidity);
  humidityFloat=(humidity*100)-(humidityInt*100);

  //get the date into memroy
  bufData[10]=humidityInt;
  bufData[11]=humidityFloat;

  return humidity;
}

/*****************************************
 * Function name: bind2bcd(byte)
 * type: byte
 * input parameters: byte val
 * output parameter:  val + 6 * (val / 10);
 * Description:
 * Creators: Jeelabs.
 *****************************************
 */
static byte bin2bcd (byte val)
{
  return val + 6 * (val / 10);
  delay(1);
}

/*****************************************
 * Function name: bcd2bcd(byte)
 * type: byte
 * input parameters: byte val
 * output parameter: val - 6 * (val >> 4);
 * Description:
 * Creators: Jeelabs
 *****************************************
 */
static byte bcd2bin (byte val)
{
  return val - 6 * (val >> 4);
  delay(1);
}

/**************************************************************************
**************************
 * // Function name: setDate(byte,  byte, byte, byte, byte)
 * type: static void
 * input parameters: byte yy: year
 * byte mm:month
 * byte dd: day
 * byte h: hour
 * byte m: minutes
 * byte s: seconds
 * output parameter: NULL
 * Description:  This function is used in order to setup a specific day in
the format yy/mm/dd h:m:s
 * Compile it once in order to set up the clock. Erase the call of this
function from the code.
```

```
 * Finally compile again in order to make the clock to start running
indepedently.
 * Creators: Jeelabs

**************************************************************************
******************************
 */
static void setDate (byte yy, byte mm, byte dd, byte h, byte m, byte s)
{
  rtc.send();
  rtc.write(0);
  rtc.write(bin2bcd(s));
  rtc.write(bin2bcd(m));
  rtc.write(bin2bcd(h));
  rtc.write(bin2bcd(0));
  rtc.write(bin2bcd(dd));
  rtc.write(bin2bcd(mm));
  rtc.write(bin2bcd(yy));
  rtc.write(0);
  rtc.stop();
}

/******************************************************************
 * Function name: getDate (void)
 * type: static void
 * input parameters: bye* buf: byte array in which the date and time would
be introduced
 * output parameter: NULL
 * Description: get the date and time from the clock and puts it into
 * the byte array specified
 * Creators: MBARI
 *****************************************************************/

static void getDate (byte* buf)
{
  rtc.send();
  rtc.write(0);
  rtc.stop();

  rtc.receive();
  buf[5] = bcd2bin(rtc.read(0));
  buf[4] = bcd2bin(rtc.read(0));
  buf[3] = bcd2bin(rtc.read(0));
  rtc.read(0);
  buf[2] = bcd2bin(rtc.read(0));
  buf[1] = bcd2bin(rtc.read(0));
  buf[0] = bcd2bin(rtc.read(1));
  rtc.stop();
}


/******************************************************************
 * Function name:saved(char[])
 * type: void
 * input parameters: char a[]= array whcih will be saved into memory
 * output parameter: NULL
 * Description: saves the input array into the memory. The way it works is
 * by checking the directory of the memory which is at location
 * [0,0] if there is nothing in memory start saving on adress [1,0]
 * if there is something on memory start saving where you left off
 * Creators: MBARI
 ******************************************************************/

void saved(char a[])
{
  int remember[2];
  if(mem.isPresent())
  {
    mem.load(0,0,remember,4);
    delay(10);
    if(remember[0]==0 && remember[1]==0)
    {
      mem.save(1,0, a,12);
      delay(10);
      remember[0]=1;
      remember[1]+=12;
      mem.save(0,0,remember,4);
      delay(10);
    }
    else
    {
      if(remember[1]<243)
      {
        mem.save(remember[0],remember[1],a,12);
        delay(10);
        remember[1]+=12;
      }
      if (remember[1]>243)
      {
        remember[0]+=1;
        remember[1]=0;
        mem.save(remember[0],remember[1],a,12);
        remember[1]+=12;
      }
      mem.save(0,0,remember,4);
      delay(10);
    }
  }
}



/******************************************************************
 * Function name: send_Radio( char a[], int length)
 * type: void
 * input parameters: char a[]: array which will be transmitted
 * int length: length of the array to be transmitted
 * output parameter: NULL
 * Description: sends an array of n length through radio communications
 * to a base station
 * Creators: MBARI
 ****************************************************************/
void send_Radio(char a[], int length)
{
  delay(10);
  while (!rf12_canSend()) // wait until sending is allowed
  {
    rf12_recvDone();    // processes any incoming data, it all happens "in
the background"
  }
  rf12_sendStart(1,a, length);  // sends the data out
}

/**************************************************************
 * Function name: transform(char [])
 * type: void
 * input parameters: char a[]: array of n bytes which will be loaded
 * output parameter: NULL
 * Description: Transform an arry of characters into something that looks
 * like mm//dd//yy h:m:s CO2 ppm= xxx temperature= xx.xx
 * humidity= xx.x
 * Creators: MBARI
 ****************************************************************/
void transform( char a[])
{
  int buffer[11];
  char string[80];
  for(int i=0;i<6;i++) //get the time form the clock
  {
    buffer[i]= (int)a[i];
  }
  buffer[6]=((a[6] &0xff)<<8) | (a[7] &0xff); //merge the two bytes of co2

  for(int i=8;i<12;i++) // get the values for temperature and humidity
  {
    buffer[i-1]=int(a[i]);
  }

  sprintf(string,"%02d/%02d/%02d %02d:%02d:%02d  CO2 ppm= %02d
temperature= %02d.%02d humidity= %02d.%02d",

buffer[0],buffer[1],buffer[2],buffer[3],buffer[4],buffer[5],buffer[6],buff
er[7],buffer[8],buffer[9],buffer[10]);

  Serial.print(string);
  Serial.println();
  delay(1);
}

/**************************************************************
 * Function name: Telaire-IdleMode(boolean mode)
 * type: void
 * input parameters: boolean mode,
 * output parameter: NULL
 * Description: Indicates wheter the sensor is
                going to go in Idle mode or not. True=Idle mode,
                False=wake up
 * Creators: MBARI
 *********************************************************************/

void Telaire_IdleMode(boolean mode)
{
  boolean out=true;
  byte buf[3];

  //if true the Telaire will go into sleep mode
  if(mode==true)
  {
    while(out)
    {
      int i=0;
      uart.write(0xFF);
      uart.write(0xFE);
      uart.write(0x02);
      uart.write (0xB9);
      uart.write (0X01);
      while(uart.available())
      {
        buf[i]=uart.read();
        i++;
      }
      if((buf[0]==255)&&(buf[1]==250)&&(buf[2]==0))
      {
        out=false;
      }
    }
  }

  //if not true, the Telair will wake up
  if(mode==false)
  {
    while(out)
    {
      int i=0;
      uart.write(0xFF);
      uart.write(0xFE);
      uart.write(0x02);
      uart.write (0xB9);
      uart.write (0X02);
      while(uart.available())
      {
        buf[i]=uart.read();
        i++;
      }
      if((buf[0]==255)&&(buf[1]==250)&&(buf[2]==0))
      {
        out=false;
      }
    }
  }
}
```

- **Appendix C**

```
//uncomment the following line to calibrate the sensor
//Calibrate_Telair();
}

void loop()
{
  //read CO2 every 10 seconds
  int co2=read_Co2();
  Serial.print("CO2 ppm=");
  Serial.println(co2);
  delay(10000);
}


/*******************************************************************
 Function name: read_Co2(void)
 type: int
 input parameters: void
 output parameter: int
 Description: tells tehe sensor to start doing measurments and store
             the two bytes in a varible called co2value
 Creators: MBARI
********************************************************************/
int read_Co2()
{
  byte buf[56];
  byte i=0;
  int co2Value;

  uart.write(0xFF);
  uart.write(0xFE);
  uart.write(0x02);
  uart.write(0x02);
  uart.write(0x03);
  delay(20);

  // put he two bit answer from the sensor into a buffer
  while(uart.available())
  {
    buf[i]=uart.read();
    i++;
  }

  // do a bitwise operation to acomodate the two bits
  // and make them an integer
  co2Value=0;
  co2Value|=buf[3] & 0xFF;
  co2Value=co2Value<<8;
  co2Value|=buf[4] & 0xFF;
  return co2Value;
```

```
}

/*****************************************************************
 Function name: Abc_Logic_Off()
 type: void
 input parameters: NULL
 output parameter: NULL
 Description:Tells the sensor to turn off the ABC logic (self calibration
             process)
 Creators: MBARI
******************************************************************/
void Abc_Logic_Off()
{
  byte buf[10]={};
  boolean out =true;
    while(out)
    {
      int i=0;
      uart.write(0xFF);
      uart.write(0xFE);
      uart.write(0x02);
      uart.write(0xB7);
      uart.write(0x02);
      delay(20);

      // put he two bit answer from the sensor into a buffer
      while(uart.available())
      {
      uart.read();
      }
      delay(20);
      //check if the abc logic was turned of
      uart.write(0xFF);
      uart.write(0xFE);
      uart.write(0x02);
      uart.write(0xB7);
      uart.write(0x00);
      delay(20);

      // put he two bit answer from the sensor into a buffer
      while(uart.available())
      {
      buf[i]=uart.read();
      i++;
      }
      Serial.println(buf[3]);
      if(buf[3]==0x01)
      {
        Serial.println("ABC logic on");
      }
      if(buf[3]==0x2)
      {
        Serial.println("ABC loigc turned off");
        out=false;
      }
      else if((buf[3]<1||buf[3]>2))
      {
        Serial.println("error reading ABC logic status");
      }
    }
}

/*******************************************************************
 Function name: Check_Abc_Logic()
 type: void
 input parameters: NULL
 output parameter: NULL
 Description:Tells the sensor chek for the ABC logic status
 Creators: MBARI
****************
```