SubNet

Final Report

REVISION HISTORY AND APPROVAL RECORD

Revision	**Date**	**Purpose**
0	22/12/23	Document creation
1	24/12/23	Document revision
2	04/01/24	Final revision

DOCUMENT DISTRIBUTION LIST

Name	**E-mail**
Pol Gonzalo Martí	pol.gonzalo@estudiantat.upc.edu
Elies García Alvira	elies.garcia.alvira@estudiantat.upc.edu
Abdelilah Korrie	abdelilah.korrie1@estudiantat.upc.edu
Aitor Pitarch	aitor.pitarch@estudiantat.upc.edu
Ignasi Fernández Bilbeny	ignasi.fernandez.bilbeny@estudiantat.upc.edu

WRITTEN BY:		REVIEWED AND APPROVED BY:	
Date	26/12/2023	Date	04/01/2024
Name	Pol Gonzalo Martí	Name	Ignasi Fernández Bilbeny
Position	Docum. Resp.	Position	Project leader

0. CONTENTS

1. DOCUMENT SCOPE

With a primary objective to build up a compact submarine equipped with some different sensors beyond the fundamental capabilities of a regular submarine, our project embarks on our principal objective which is to give support to create, supervise and maintain underwater infrastructure (e.g: submarine cables). It would also have some other important applications like checking a few water quality KPI's (e.g: pH, Electrical conductivity, total dissolved solids…).

In our relentless pursuit of a tangible Remote Operated Underwater Vehicle (ROUV), we have completed the goal through diverse mission stages, meticulously mentioned on this complete report. If we do a general overview of the project we can realize that it is basically splitted in two main parts, Part A and Part B.

Part A predominantly revolves across the foundational construction of the submarine. This involves the meticulous fabrication of the physical shape of the ROUV, complemented by the integration of three superior propellers. After the physical part is done, we have to complete the basic electronics and software part to make possible the gathering of data of the pressure sensor and the mobility of the ROUV by using three different switches (One for each propeller).

Conversely, Part B is more focused on upgrading the initial version of the ROUV. Primarily targeted at the electronic field, this section includes the precise implementation of sensors and drivers onto the board, making sure seamless integration. Simultaneously, the software program aspect of Part B delves into getting the control of the submarine's mobility by using Arduino and the drivers.

As we mentioned before, the project has two main parts:

PART A	PART B
WP1: Mechanical	WP6: Final assembling and water test
WP2: Electrical	WP7: Temperature sensor
WP3: Electronics	WP8: Drivers
WP4: Communication	WP9: Turbidity & TDS sensor
WP5: Signal Processing	WP10: Graphical user interface
	WP11: Hardware integration

2. TIME PLAN UPDATED

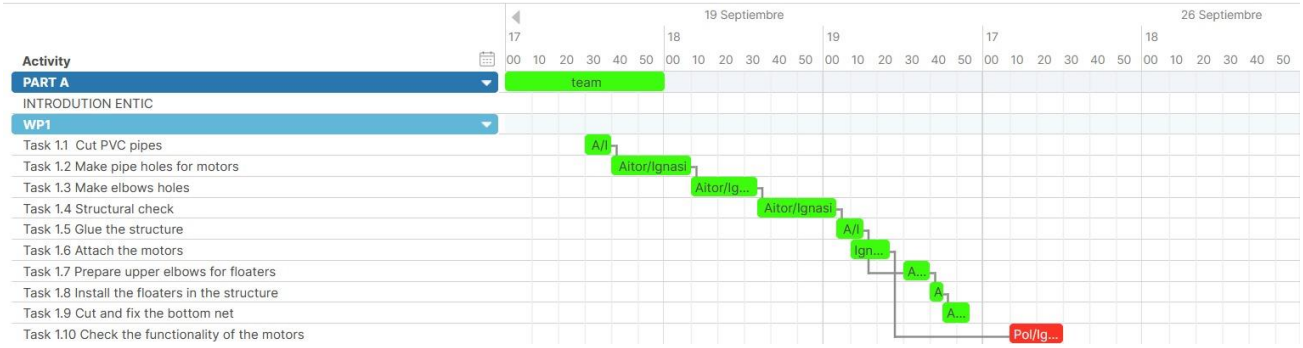
Before starting our project, we knew it was necessary to plan the project considering the time available and to distribute the work across sessions, setting limits to finish each task. To elaborate this time planning we used Gantt and as the sessions ended we wrote down in Gantt if we accomplished the tasks within the time limit.

PART A

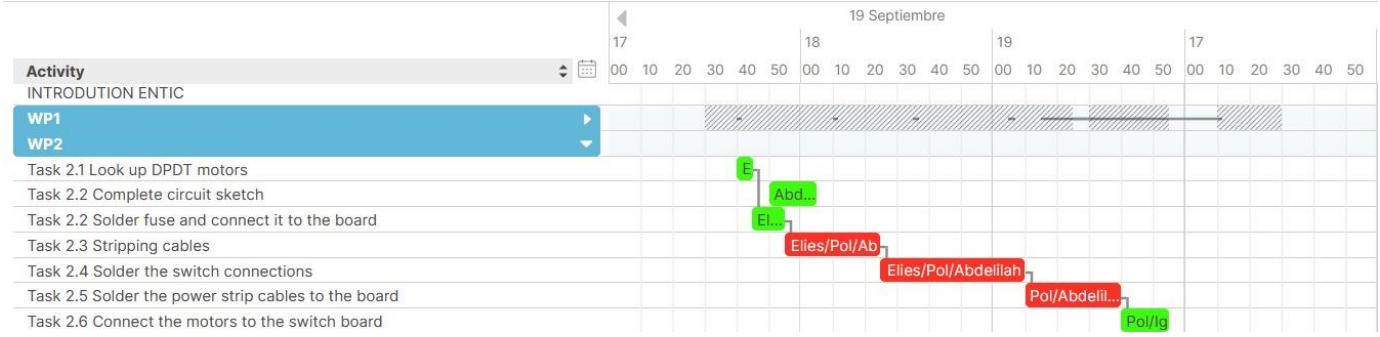
In the first session we decided to do one work package per session as we had 6 sessions and 6 work packages. If there was any delay it had to be corrected as another task was made.

At the end of session 1 we learned that we could do more than one work package and that each work package had a different time length.

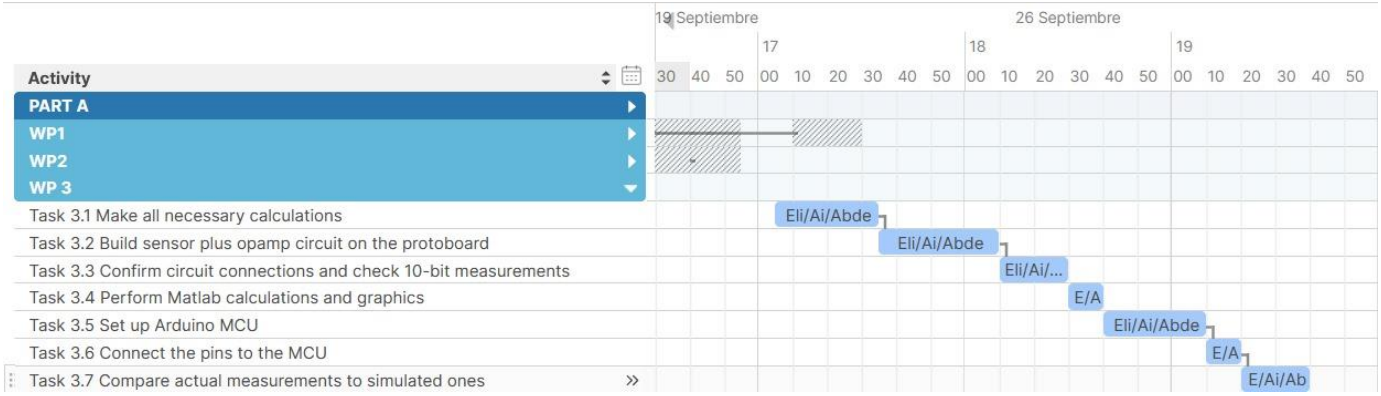
WP1:



WP 2:

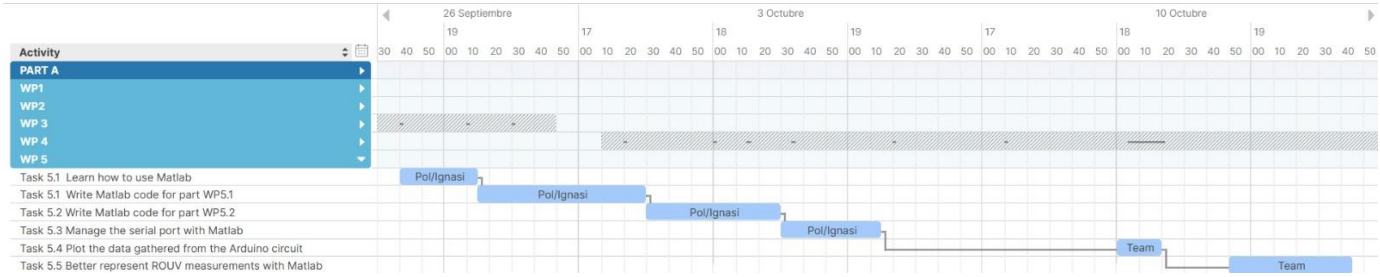


WP 3:



WP4:

WP 5:

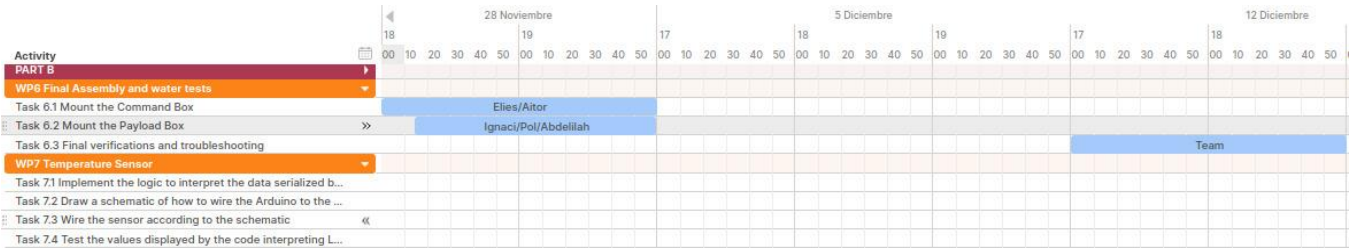


At the beginning, as we can see by the color red in different tasks, we had some delays on WP1 and WP2 which we solved in the next sessions. Once we solved those delays, we followed the schedule planned every week. The first session was decisive to understand how we needed to distribute the work for the next sessions. The blue color in the whole work package indicates that we ended all of its tasks in time. If we look at the whole part A gantt with perspective we conclude that despite bad planning in the beginning, we adapted and achieved our objective.

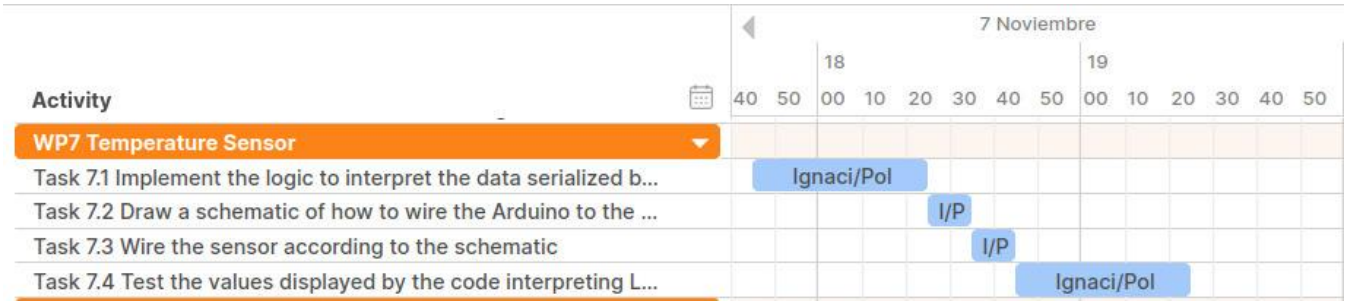
PART B

For part B, we had much more time to elaborate a plan to distribute the work across the time. We only planned when every work package had to be ended to subsequently elaborate a gantt so we could see exactly when we ended any task. So once we had part B distributed by sessions it was essential to stick to the schedule because we didn't plan to have any extra sessions to correct possible delays.

WP 6:



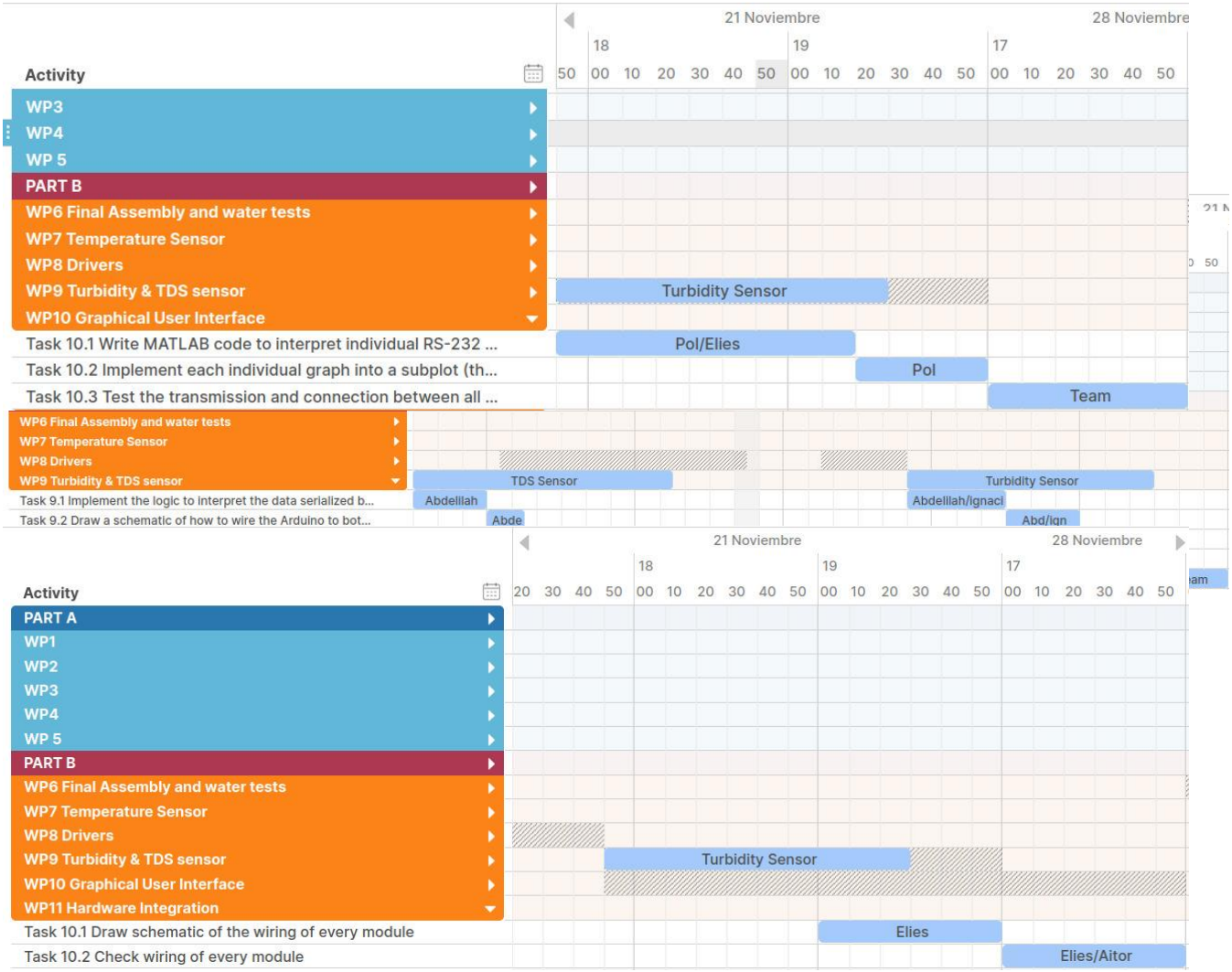
WP 7:



WP 8:

WP 9:

WP 10:

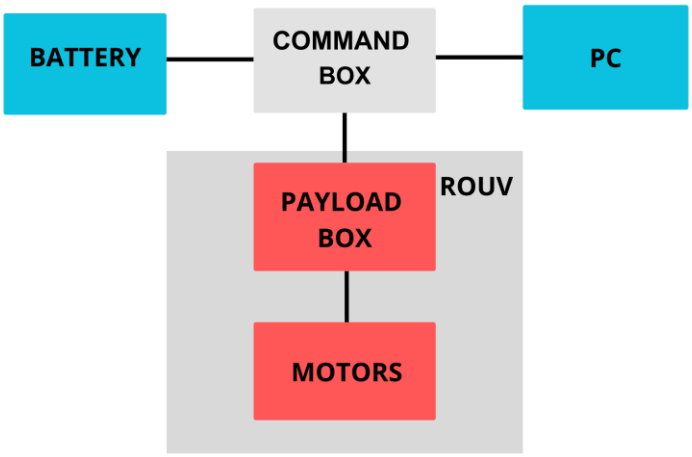


WP 11:

We achieved our objective to stick to the schedule and finish all the tasks and consequently, the project. It would have been good to keep the last session free to prevent last minute delays, in consequence we would have assigned more work in the other sessions which could have caused delays. Either way, we think we planned a schedule which was demanding and possible at the same time.

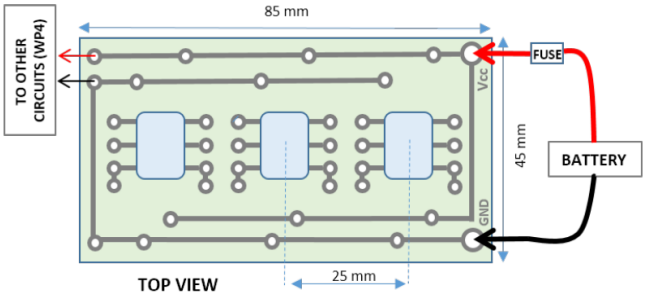
3. SYSTEM DESIGN & IMPLEMENTATION DOCUMENTATION

- **System block diagram**

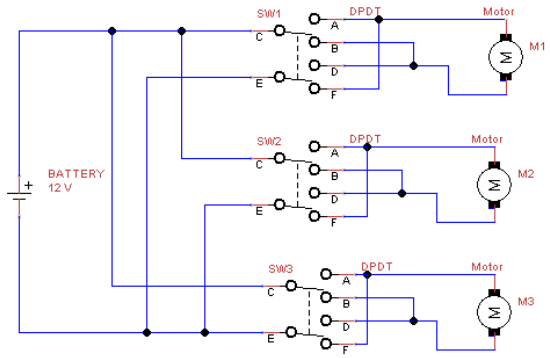


- Command box:

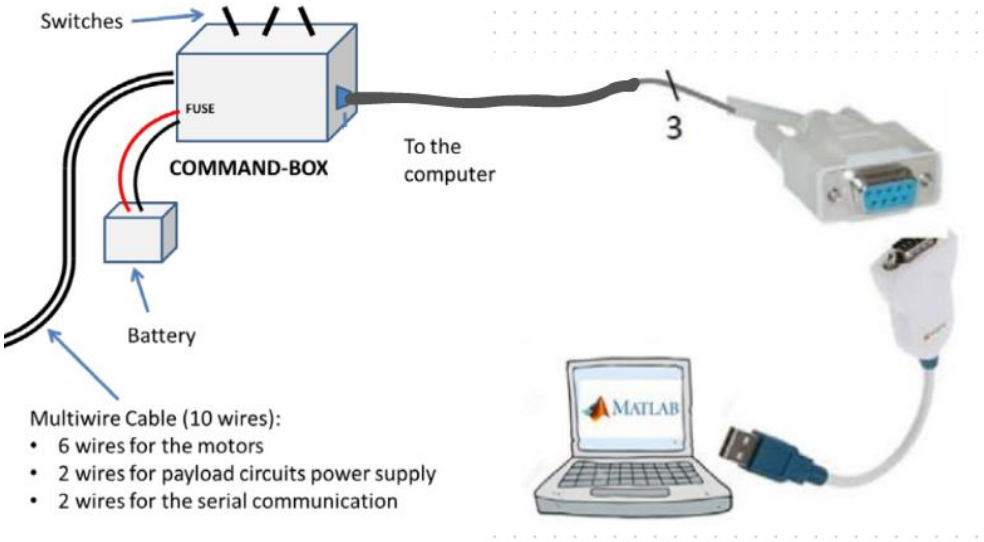
The ROUV is managed by the command box, which houses three switches for motor control, communication port, battery power supply cables, and the link to the ROUV. Within the box, essential electrical connections are present to regulate the vehicle, featuring a fuse for electrical system protection and a DB9 connector for serial communication.



Switches PCB power supply connections

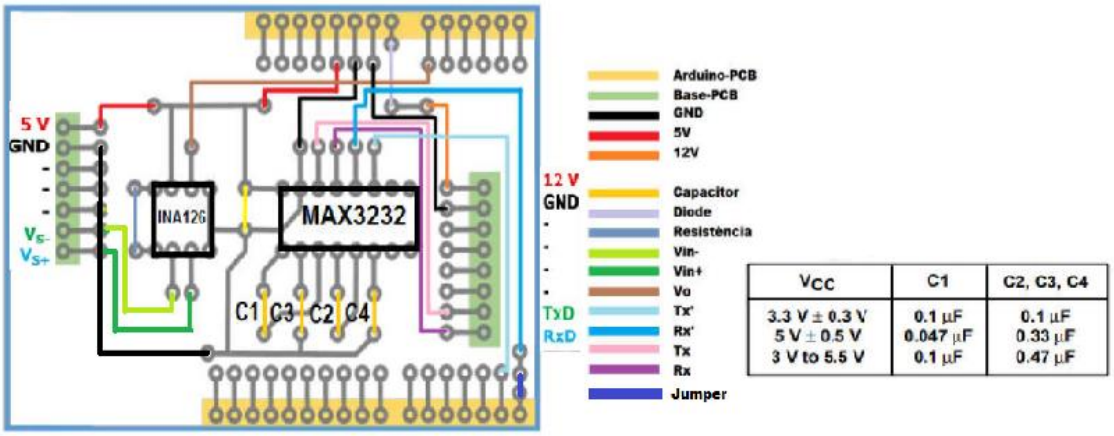


Motor switches



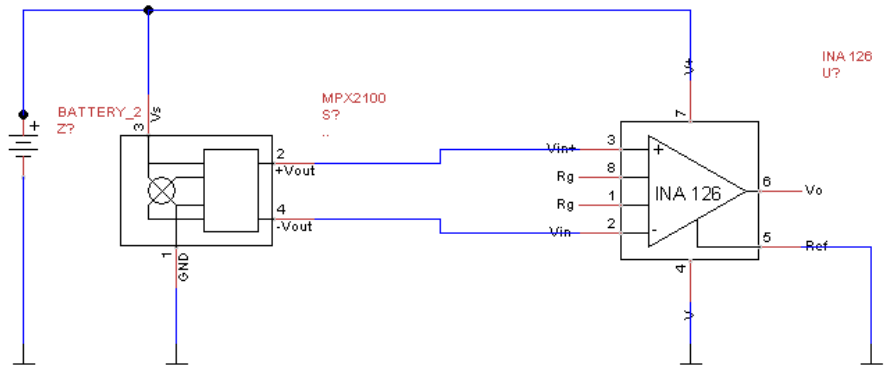
Payload box:

In Part A of the project, the payload box contains an Arduino, a pressure sensor, all connected to the PCB base.

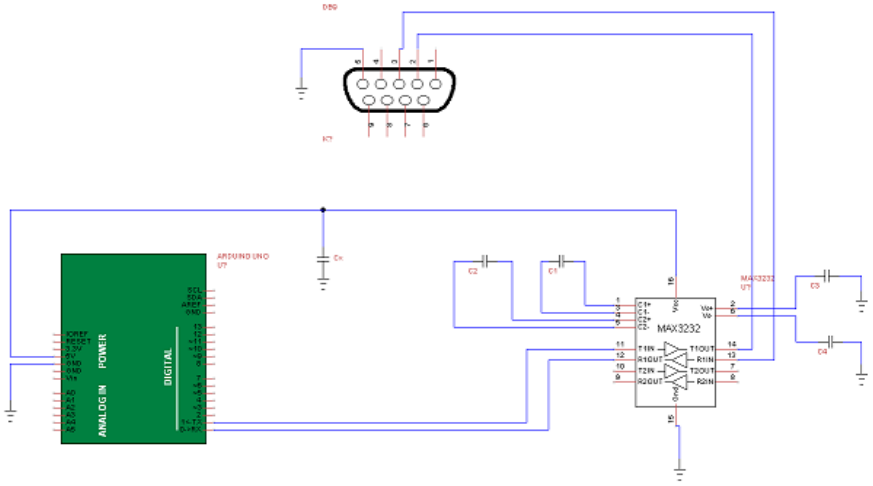


PCB base

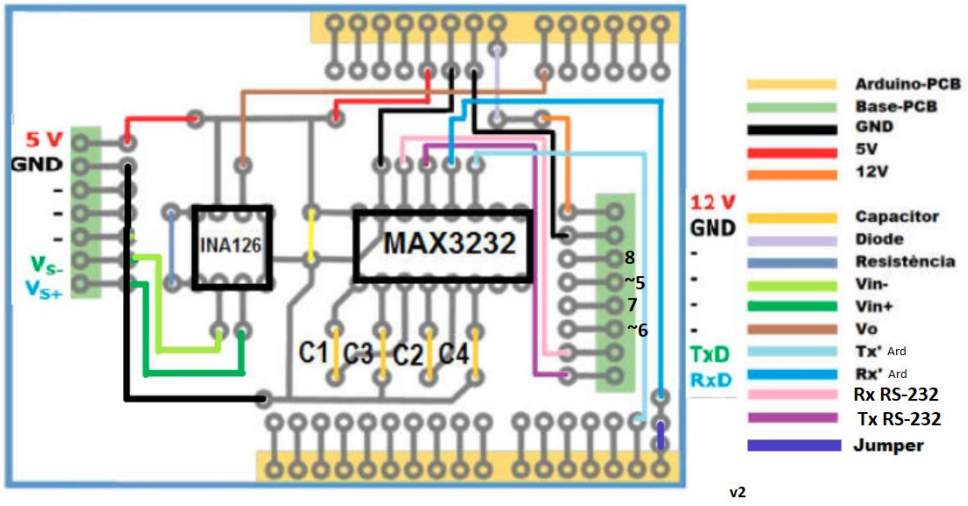
INA126, converts the voltage of the sensor to higher values.



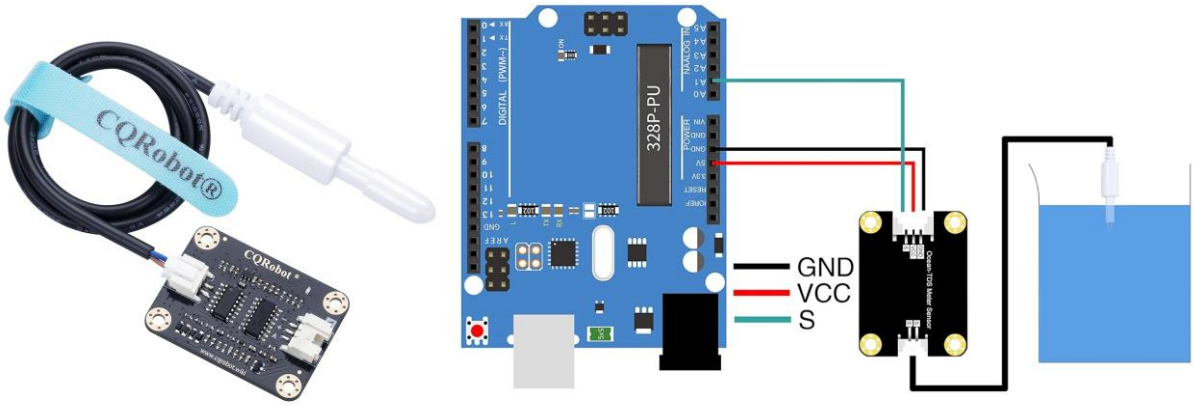
MAX3232 make the signal understandable to the Arduino.



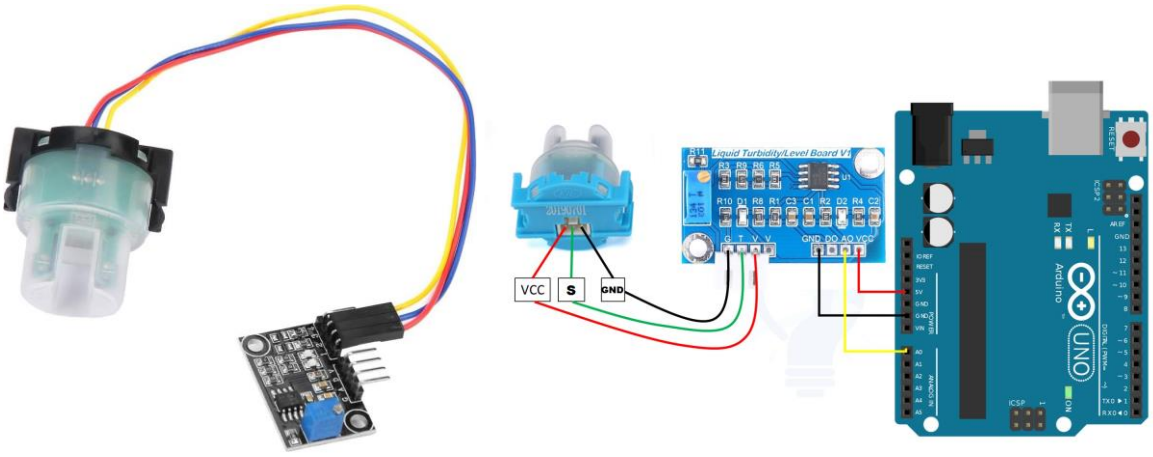
In Part B of the project, which is more advanced and features additional specifications, the payload box includes an Arduino,motor drivers, a pressure sensor, a Turbidity Sensor, a temperature sensor, and a TDS sensor, all of which are connected to the PCB base.

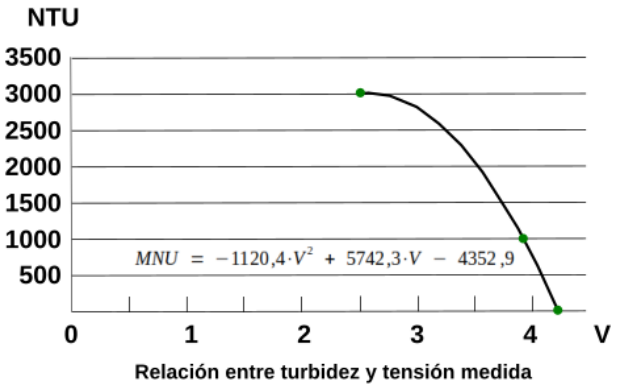


TDS sensor: The Total Dissolved Solids (TDS) sensor measures the concentration of dissolved solids in a liquid. It provides a numerical value representing the total amount of dissolved substances, including salts, minerals, and other organic and inorganic compounds, in water. TDS measured in parts per million (**ppm**)



Turbidity Sensor: A Turbidity Sensor measures the cloudiness or haziness of a fluid caused by suspended particles that are not dissolved. It quantifies the relative clarity of a liquid by detecting the scattering and absorption of light as it passes through the liquid. Turbidity is often caused by particles such as silt, clay, algae, or other microscopic organisms. The measurement is usually expressed in Nephelometric Turbidity Units (**NTU**) .



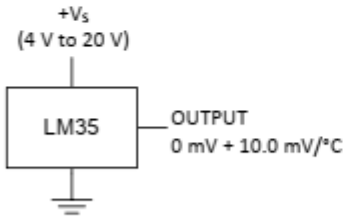


Relación entre turbidez y tensión medida

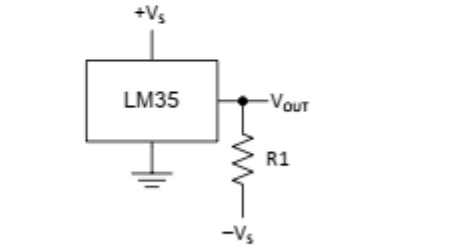
Temperature sensor LM35: The sensor produces a voltage output that is directly proportional to the temperature in degrees Celsius. The LM35 device achieves typical accuracies of ±¼°C at room temperature and ±¾°C across a complete temperature range from -55°C to 150°C without the need for external calibration or trimming.

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LM35	TO-CAN (3)	4.699 mm × 4.699 mm
	TO-92 (3)	4.30 mm × 4.30 mm
	SOIC (8)	4.90 mm × 3.91 mm
	TO-220 (3)	14.986 mm × 10.16 mm

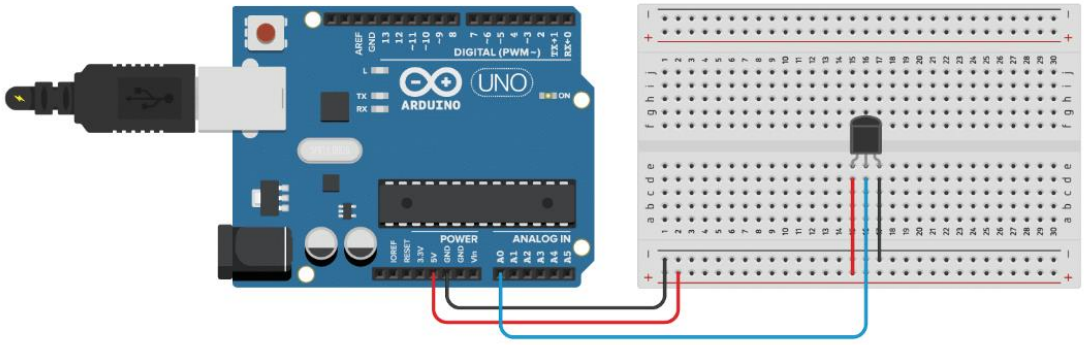
Basic Centigrade Temperature Sensor (2°C to 150°C)



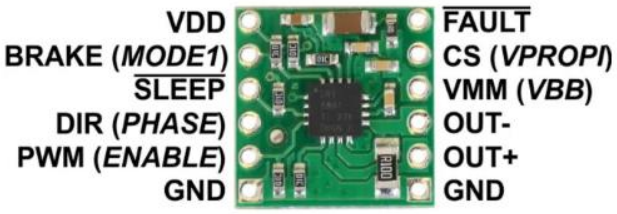
Full-Range Centigrade Temperature Sensor

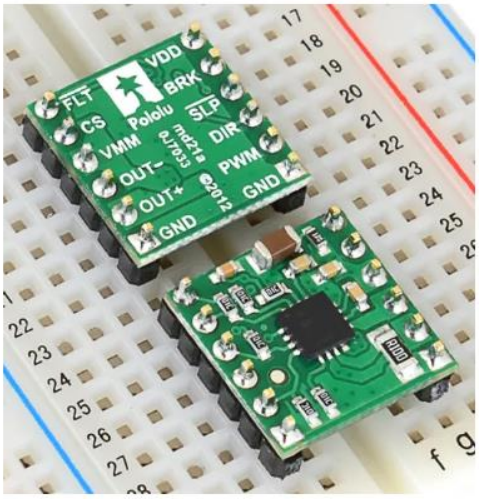


Choose $R_1 = -V_S / 50\ \mu A$
$V_{OUT} = 1500$ mV at 150°C
$V_{OUT} = 250$ mV at 25°C
$V_{OUT} = -550$ mV at –55°C

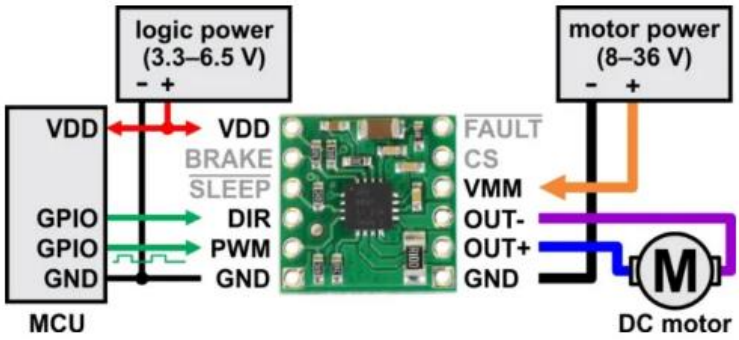


Motor drivers: A motor controller is an electronic circuit that controls the speed and direction of a motor. It takes input signals from a controller, such as a microcontroller or microprocessor, in our case the Arduino board, and translates them into electrical signals that control the motor's power supply. We have used 3 drivers, each of them controls one of the three motors.





The driver we have used is a single channel H-bridge motor driver designed to control brushed DC motors with a voltage range of 8V to 36V. It can provide up to 1A continuous current and 2.8A of maximum current, making it suitable for driving small to medium sized motors.



Code:

To allow for multitasking, we've designed the program to use the millis() function instead of the delay() built-in method. This allows for the code to precisely use the TC0 counter linked to the main Arduino registers and not waste time waiting for the delay function.

This can be seen in the code

```
if (millis() - analogSampleTimepoint > sampleRate) // every 20 milliseconds, read the analog value from the ADC
```

To be able to test the code extensively, we've implemented a "debug" mode that adds ~1,5 KB to the flash memory (which is around 5% of total Arduino memory) which helps us debug any issues quickly by adding terminal serial print statements which in Arduino occupy a substantial amount of memory.

WITHOUT DEBUG MODE (5736 bytes)

```
Compiling .pio/build/uno/FrameworkArduino/wiring_pulse.c.o
Compiling                                 kArduino/wiring_shift.c.o
Archiving                                 vorkArduino.a
Indexing .pio/build/uno/libFrameworkArduino.a
Linking .pio/build/uno/firmware.elf
Checking size .pio/build/uno/firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM:   [==        ]  19.3% (used 395 bytes from 2048 bytes)
Flash: [==        ]  17.8% (used 5736 bytes from 32256 bytes)
Building .pio/build/uno/firmware.hex
========================================== [SUCCESS] Took 3.96 seconds
```

Open file in editor (ctrl + click)

WITH DEBUG MODE (7216 bytes)

```
Scanning dependencies...
No dependencies
Building in release mode
Compiling .pio/build/uno/src/main.cpp.o
Linking .pio/build/uno/firmware.elf
Checking size .pio/build/uno/firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM:   [=====     ]  49.2% (used 1008 bytes from 2048 bytes)
Flash: [==        ]  22.4% (used 7216 bytes from 32256 bytes)
Building .pio/build/uno/firmware.hex
========================================== [SUCCESS] Took 1.90 seconds
```

The sensor readings are batched in 20 ms increments and after a certain set sample rate they're averaged using a simple averaging filter (except for the TDS readings which are filtered using a median filter).

4. SYSTEM SPECIFICATIONS

In the early stages, the idea involved creating a remotely operated underwater vehicle controlled through a command box with three switches. This external control unit was linked to the submarine's operational module via a cable spanning ten meters.

The primary goal of the underwater vehicle was to methodically gather information related to temperature, pressure, turbidity and TDS (Total Dissolved Solids) levels as it navigated along a predetermined pathway. Following this, the obtained data is sent to a computing system for processing and visualization. The submarine incorporates an automated navigation feature to maintain its designated course.

The sensors and drivers were installed after assembling the structure and tested as a set. The drivers are one channel, so each was attached to a different motor.

The sensor data gathered at the Arduino is sent serially to the computer at the surface through the umbilical cable. Plots show real-time data and at the same time offer computer control of the motors through captain mode in the program code (see section 9).

All in all, the ROUV is a trimotor submarine vehicle equipped with the aforementioned sensors, computer motor control hardware including the drivers and Arduino UNO microcontroller. Specifically, the sensors are the following:

The LM35 temperature sensor mounted on the perforated board, along with the TSW turbidity sensor and the TDS total dissolved solids module.

On the other side of the board is the MPX pressure sensor, along with the drivers and their sockets.

The software is designed to send a frame containing the sensor data following a format where each data is separated by a white space and the frame itself is terminated with a CR/LF character.

Simultaneously, graphs are updated every 9–12 milliseconds (80–110 fps). There are four graphs located on a square 2x2 grid and axes are updated when data surpasses an axis limit.

Computer motor control is possible using the keyboard (or some other peripheral) and sending through the serial link certain combinations of characters detailed in the manual. This is interpreted in the code with a switch case statement and using a function to simplify directing each motor individually.

Additionally, data treatment is done directly in the microcontroller with a simple averaging filter. Except the TDS data, which goes through a median filter to improve data robustness.

5. SYSTEM OPERATION DOCUMENTATION

The ROUV is controlled by transmitting characters in succession through the serial link, following this scheme:

- Captain mode (Electronically drive ROUV with drivers)
    - Uses keyboard layout for duty cycle percentages (first character that has to be sent)
        - a-q-w-e-r (0, 25, 50, 75, 100)% left motor
        - s-f-t-y-h (0, 25, 50, 75, 100)% center motor
        - d-u-i-o-p (0, 25, 50, 75, 100)% right motor
    - Turn control (second character that can be sent, default is clockwise)
        - k-j (clockwise, anticlockwise)
    - Autopilot (replay saved commands in buffer)
        - g (save commands from now on)
        - “ “ (whitespace character stops recording commands)
        - h (replay saved commands in buffer)

Image with colored commands (blue is left motor, orange is center motor, green is right motor). The a-s-d keys are stop commands for the left, center and right motors respectively.

Examples:

- q-k => left motor @ 25% clockwise
- j-j => center motor @ 100% anticlockwise
- a => left motor @ 0% (stops)
- o-k => right motor @ 75% clockwise
- r => left motor @ 100% clockwise (default)

6. SYSTEM COST

MATERIALS AND COMPONENTS

Material List	Price per unit (€)	Number of units	Total price (€)
3 motors set	69	1	69
Clamps	0,8	3	2,4
Screws	0,2	3	0,6
Washers	0,1	6	0,6
Power Supply wires	0,5	1	0,5
Link cable	9,4	1	9,4
Floaters	0,8	2	1,6
Sealing gland	1,5	3	4,5
Rubber O-ring	0,3	3	0,9
DB9 Serial	0,94	2	1,88
DB9 screws & nuts	0,2	2	0,4
Fuse clip	1,02	1	1,02
Fuse	0,3	1	0,3
Payload box fixing	3,06	4	12,24
Box fixing screws	0,1	4	0,4
Electrical connector	0,7	1	0,7
PCB jumper	0,97	1	0,97
Pressure sensor	10,98	1	10,98
Differential amplifier	2,87	1	2,87
RS232 transceiver	1,06	1	1,06

Arduino UNO	17	1	17
PVC pipe (1m)	1,65	2	3,3
PVC elbow	0,4	10	4
PVC tee	0,6	4	2,4
Floater support (50 cm)	0,1	1	0,1
Plastic net (40x30 cm)	0,3	1	0,3
Command box	5,8	1	5,8
Switches	2,3	3	6,9
Payload box	14,78	1	14,78
Silicon pipe (10 cm)	0,3	1	0,3
Pipe terminal	0,02	1	0,02
PCB prototype board	5,78	1	5,78
PCB connector	0,5	1	0,5
Arduino prototype PCB	3	1	3
8 pins IC socket	0,3	1	0,3
16 pins IC socket	0,31	1	0,31

Capacitors	0,15	5	0,75
PCB socket strip	3,14	2	6,28
Short PCB pins strip	0,2	1	0,2
Long PCB pins strip	0,68	1	0,68
Diode	0,04	1	0,04
Plastic strip	0,02	15	0,3
Rigid connection wires	0,2	10	2
Flexible connection wires	0,15	1	0,15
Turbidity sensor	19,77	1	19,77
TDS sensor	16,98	1	16,98
Drivers Pololu md21a	11,73 (12,95 USD)	1	35,19
LM 35	1,5	1	1,5
TOTAL		**270,95**	

NUMBER OF HOURS-PERSON FOR EACH MAIN TASK

WP (PART A: 1-5 & PART B: 6-11)	Ignasi Fernández	Abdelilah Korrie	Aitor Pitarch	Elies Garcia	Pol Gonzalo
WP 1	2h 50 min	1h 30 min	2h 30 min	–	20 min
WP 2	20 min	2h	–	1h 30 min	1h 45 min
WP 3	–	2h 35 min	2h 35 min	2h 35 min	–
WP 4	2h 10 min	3h 30 min	3h 30 min	4h 10 min	2h 10 min
WP 5	4h 10 min	1h 10 min	1h 10 min	1h 10 min	4h 10 min
WP 6	3h	3h	3h	3h	3h
WP 7	1h 30 min	–	–	–	1h 30 min
WP 8	–	–	2h 15 min	2h 15 min	25 min
WP 9	3h 30 min	3h 30 min	30 min	30 min	30 min
WP 10	1h	1h	1h	2h 30 min	3h 10 min
WP 11	2h	–	2h	2h	–

7. CONCLUSIONS

In conclusion, our expedition with Project SubNet has been nothing short of a captivating and transformative odyssey. While the ultimate aim of submerging the ROUV underwater remained unfulfilled, the insights garnered from this endeavor are priceless treasures that will undoubtedly illuminate our paths as future telecommunications engineers.

With the gift of hindsight, we recognize that this journey has been a masterclass, providing profound lessons across all facets of our engineering education over the two initial years. It's an exploration that surpassed our expectations, allowing us to implement knowledge that we never thought possible. This project served as a canvas for us to unleash our fullest potential, delving into new realms of understanding autonomously and working independently to propel the project forward.

As we reflect on our journey, the profound impact of our professors' unwavering guidance emerges as a cornerstone of our success. Their dedication not only facilitated the technical aspects of the project but also enriched our learning experience. The collaborative ethos within our team, marked by effective communication, optimism, and enthusiasm, created an environment conducive to innovation. However, the journey also unveiled areas beckoning improvement, including the cultivation of composure in the face of unexpected challenges and the strategic allocation of time.

Project SubNet, beyond being a reservoir of technical knowledge, has been a crucible for refining our abilities in teamwork, adaptability, and responsibility. The realization that setbacks can coexist with dedicated effort and meticulous planning has added a nuanced layer to our professional growth.

As we bring this project to a close, we carry with us not only a wealth of experiences but also a renewed sense of purpose and resilience. The echoes of this undertaking will resonate in our future endeavors in engineering and collaborative initiatives. Despite the unfulfilled dream of witnessing the ROUV submerged underwater, the journey itself has become the destination, brimming with lessons, growth, and the indomitable spirit of exploration. In the end, our yearning to see the project's completion remains, an eternal flame that fuels our passion for future challenges and triumphs in the ever-evolving landscape of engineering.

8. REFLECTION DOCUMENT

Our professors have played an instrumental role in the success of our project, providing unwavering support and guidance both in theory and in the laboratory. Their expertise and responsiveness to our queries were crucial, and we deeply appreciate the knowledge they shared. We genuinely believe they excelled in their roles, and we are grateful for the invaluable contribution they made to our project.

As a team, our communication, positivity, and enthusiasm for the project were strong attributes. However, reflecting on our performance, we acknowledge an area for potential improvement: maintaining composure in the face of unexpected challenges. The delay of over 6 hours due to an unforeseen problem was a setback that affected team morale. In retrospect, remaining calm during such situations could have facilitated a more efficient resolution.

Our teamwork was indeed gratifying, characterized by effective conflict management and collaboration. Despite the majority of the team members being unfamiliar with each other, communication was seamless. When challenges arose, team members readily provided support and solutions, fostering a respectful and cohesive environment.

In terms of self-assessment, each team member brought valuable skills to the project. However, recognizing the need for enhanced composure in challenging situations is crucial. Additionally, proactive problem-solving and a more structured approach to handling unexpected issues could further improve future project outcomes.

Looking back, we realize that allocating more time to certain aspects of the project might have been beneficial. Additionally, acknowledging and following more closely the opinions and guidance of our lab teacher could have steered us toward a smoother project execution. Embracing these lessons will undoubtedly contribute to our growth and success in future endeavors.

This project served as a transformative journey, teaching us the importance of collaboration with diverse individuals and confident communication in public settings. The emphasis on responsibility regarding project deadlines was a key lesson, enabling us to navigate challenges more effectively. Overall, the experience was enriching, allowing us to find joy in the process while gaining valuable insights into teamwork and project management.

9. ANNEXES

Presented in this section is the complete software suite and all the datasheets of our ROUV.

The complete software code can be found in the following GitHub repository:

https://github.com/alias313/ROUV

It's meant to be executed using the PlatformIO VSCode plugin (VSCode isn't absolutely necessary but recommended. Without VSCode PlatformIO can also be executed on a terminal application).

The /src folder contains the main program that the microcontroller runs named main.cpp which at compile time joins the main.hpp and debug.hpp header files.

The /graphs folder contains two different ways to represent data, the one that was tested in the end was the matlab file ScriptENTICMatlab.m

Main.cpp

```
#include <Arduino.h>
#include "main.hpp"
#include "debug.hpp"

const uint8_t MPXPin = A0;   // MPX2100AP (pressure in kPa) sensor
const uint8_t TDSPin = A3;   // TDS (Total Dissolved Solids in ppm) sensor
const uint8_t TSWPin = A2;   // TSW-20M Turbidity (Nephelometer in NTU) sensor
const uint8_t LMPin = A1;   // LM35 temperature sensor

/* Left motor */
const uint8_t DIRL = 2;
const uint8_t PWML = 3;

/* Center motor */
const uint8_t DIRC = 12;
const uint8_t PWMC = 10;

/* Right motor */
const uint8_t DIRR = 13;
const uint8_t PWMR = 11;

void turnMotor(char motor, char turnDirection, int dutyCycle);
int getMedianNum(int bArray[], int iFilterLen);
void verboseMessages();

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

   pinMode(MPXPin, INPUT);
   pinMode(TDSPin, INPUT);
   pinMode(TSWPin, INPUT);

   pinMode(DIRL, LOW);
   pinMode(PWML, LOW);
   pinMode(DIRC, LOW);
   pinMode(PWMC, LOW);
```

```
    pinMode(DIRR, LOW);
    pinMode(PWMR, LOW);

    // while (debugAvailable() <= 0 || debugRead() != 's')
    // {
    //     debugln("Send 's' to start: ");
    //     debugDelay(200);
    // }
    debugln("Lectura de: presion (manometro), turbidez (turbidimetro), tds (conductimetro) y velocidad");
    debugln("Conversion a medidas utiles: tension => profundidad, turbidez, total de solidos disueltos");
    debugln("=============================================================================================");
    debugln(" ");
    debugln("MPX: Tension\tTDS: Tension / TDS   \tTSW: Tension / NTU   ");
    debugln("------------\t--------------------\t--------------------");
}

void loop()
{
    while (Serial.available() > 0)
    {
        char inputByte = Serial.read();
        debug("Serial is available with byte '");
        debug(inputByte);
        debugln("' read");

        if (inputByte != '\n')
        {
            command[command_pos] = inputByte;
            debug("Command_pos: ");
            debugln(command_pos);
            command_pos++;
        } else {
            command[0] = 'c'; // First character indicates whether captain mode is on/off, 'c' = on
            command_pos = 1;
            debugln("newline");
            debug("Characters in serial buffer left: ");
            debugln(Serial.available());
            break;
        }
        debug("Characters in serial buffer left: ");
        debugln(Serial.available());
    }

    if (command[0] == 'c')
    {
        commandKey = command[1];
        turn = command[2];
        switch (commandKey)
        {
        case 'a':
            digitalWrite(DIRL, LOW);
            analogWrite(PWML, STOP);
            break;
        case 'q':
            turnMotor('l', turn, LOW_SPEED);
            break;
        case 'w':
            turnMotor('l', turn, HALF_SPEED);
            break;
        case 'e':
```

```
        turnMotor('l', turn, HIGH_SPEED);
        break;
    case 'r':
        turnMotor('l', turn, FULL_SPEED);
        break;

    case 's':
        digitalWrite(DIRC, LOW);
        analogWrite(PWMC, STOP);
        break;
    case 'f':
        turnMotor('c', turn, LOW_SPEED);
        break;
    case 't':
        turnMotor('c', turn, HALF_SPEED);
        break;
    case 'y':
        turnMotor('c', turn, HIGH_SPEED);
        break;
    case 'h':
        turnMotor('c', turn, FULL_SPEED);
        break;

    case 'd':
        digitalWrite(DIRR, LOW);
        analogWrite(PWMR, STOP);
        break;
    case 'u':
        turnMotor('r', turn, LOW_SPEED);
        break;
    case 'i':
        turnMotor('r', turn, HALF_SPEED);
        break;
    case 'o':
        turnMotor('r', turn, HIGH_SPEED);
        break;
    case 'p':
        turnMotor('r', turn, FULL_SPEED);
        break;
    case '\r':
        debugln("\r");
        break;

    default:
        debug("Command key '");
        debug(commandKey);
        debugln("' not recognized.");
        break;
    }

    command[0] = 'n'; // captain mode off
}
static unsigned long analogSampleTimepoint = millis();
if (millis() - analogSampleTimepoint > sampleRate) // every 20 milliseconds, read the analog value from the ADC
{
    // TDS sensor
    analogSampleTimepoint = millis();
    analogBuffer[analogBufferIndex] = analogRead(TDSPin); // read the analog value and store into the buffer
    analogBufferIndex++;
    if (analogBufferIndex == SCOUNT)
```

```
            analogBufferIndex = 0;

        // LM35 temperature sensor
        lmVoltage += ((float)analogRead(LMPin) / 1024) * 5000; // mV because every 10 mV is 1 degree Celsius

        // MPX pressure ==> depth sensor
        mpxVoltage += ((float)analogRead(MPXPin) / 1024) * 5;

        // Turbitity sensor
        tswVoltage += ((float)analogRead(TSWPin) / 1024) * 5; //mapping from 10-bit voltage reading to 0-5V range
    }
    static unsigned long printTimepoint = millis();
    if (millis() - printTimepoint > window*sampleRate)
    {
        // Simple averaging for depth and temperature values
        mpxVoltage /= window;
        lmVoltage /= window;

        depth = mpxVoltage * depthCoefficient - depthOffset;
        tempValue = lmVoltage / 10;

        mpxVoltage = 0.0;
        lmVoltage = 0.0;

        // Apply median filtering algorithm to data read from TDSPin
        printTimepoint = millis();
        for (copyIndex = 0; copyIndex < SCOUNT; copyIndex++)
            analogBufferTemp[copyIndex] = analogBuffer[copyIndex];
        tdsAverageVoltage = getMedianNum(analogBufferTemp, SCOUNT) * (float)VREF / 1024.0;
        float compensationCoefficient = 1.0 + 0.02 * (tempValue - 25.0);
// temperature compensation formula: fFinalResult(25^C) = fFinalResult(current)/(1.0+0.02*(fTP-25.0));
        float compensationVoltage = tdsAverageVoltage / compensationCoefficient;
// temperature compensation
        tdsValue = (133.42 * compensationVoltage * compensationVoltage * compensationVoltage - 255.86 * compensationVoltage *
compensationVoltage + 857.39 * compensationVoltage) * 0.5; // convert voltage value to tds value

        // https://fgcoca.github.io/Sensores-actuadores-y-shield-tipo-Arduino/turbidez/
        tswVoltage /= window;
        if (tswVoltage < 2.5)
        {
            ntuValue = 3000;
        } else {
            ntuValue = -1120.4 * square(tswVoltage) + 5742.3 * tswVoltage - 4352.9;
        }
        tswVoltage = 0.0;

        // Serial debugging
        // debugMessages();

        // Serial communication (frame sent to master)
        // debug("\tData in array: ");
        Serial.print(tempValue, 1);    // 4 bytes
        Serial.print(" ");         // 1 byte
        Serial.print(depth, 1);        // 4 bytes
        Serial.print(" ");         // 1 byte
        Serial.print(tdsValue);     // 2 bytes
        Serial.print(" ");         // 1 byte
        Serial.println(ntuValue);     // 2 bytes
    }
}
```

```
void turnMotor(char motor, char turnDirection, int dutyCycle)
{
   switch (motor)
   {
   case 'l':
      if (turnDirection == CLOCKWISE_LETTER) digitalWrite(DIRL, CLOCKWISE);
      else if (turnDirection == ANTICLOCKWISE_LETTER) digitalWrite(DIRL, ANTICLOCKWISE);
      analogWrite(PWML, dutyCycle);
      break;
   case 'c':
      if (turnDirection == CLOCKWISE_LETTER) digitalWrite(DIRC, CLOCKWISE);
      else if (turnDirection == ANTICLOCKWISE_LETTER) digitalWrite(DIRC, ANTICLOCKWISE);
      analogWrite(PWMC, dutyCycle);
      break;
   case 'r':
      if (turnDirection == CLOCKWISE_LETTER) digitalWrite(DIRR, CLOCKWISE);
      else if (turnDirection ==  ANTICLOCKWISE_LETTER) digitalWrite(DIRR, ANTICLOCKWISE);
      analogWrite(PWMR, dutyCycle);
   default:
      debug("Turn key '");
      debug(turnDirection);
      debugln("' not recognized.");
      break;
   }
   debug("Command ");
   debug(motor);
   debug(turnDirection);
   debugln(dutyCycle);
}

int getMedianNum(int bArray[], int iFilterLen)
{
   int bTab[iFilterLen];
   for (byte i = 0; i < iFilterLen; i++)
      bTab[i] = bArray[i];
   int i, j, bTemp;
   for (j = 0; j < iFilterLen - 1; j++)
   {
      for (i = 0; i < iFilterLen - j - 1; i++)
      {
         if (bTab[i] > bTab[i + 1])
         {
            bTemp = bTab[i];
            bTab[i] = bTab[i + 1];
            bTab[i + 1] = bTemp;
         }
      }
   }
   if ((iFilterLen & 1) > 0)
      bTemp = bTab[(iFilterLen - 1) / 2];
   else
      bTemp = (bTab[iFilterLen / 2] + bTab[iFilterLen / 2 - 1]) / 2;
   return bTemp;
}

void verboseMessages() {
      debug("MPX: ");
      debugDec(mpxVoltage, 2);       // 4 Bytes
      debug("V ");
```

```
        // debug(sizeof(mpxVoltage));
        // debug("Bytes");
        debug("\tTDS: ");
        debugDec(tdsAverageVoltage,2);  // 4 Bytes
        debug("V / ");
        // debug(sizeof(tdsAverageVoltage));
        // debug("Bytes / ");
        debug(tdsValue);
        debug(" ppm ");
        // debug(sizeof(tdsValue));       // 2 Bytes
        // debug("Bytes");
        debug("\tTSW: ");
        debug((float)analogRead(TSWPin) / 1024 * 5);      // 2 Bytes
        debug("V / ");
        // debug(sizeof((float)analogRead(TSWPin) / 1024 * 5));
        // debug("Bytes / ");
        debug(ntuValue);              // 2 Bytes
        debug(" NTU ");
        // debug(sizeof(ntuValue));
        // debugln("Bytes");
}
```

Main.hpp

```
#define VREF 5.0      // analog reference voltage(Volt) of the ADC
#define SCOUNT  30           // sum of sample point

#define STOP 0
#define LOW_SPEED 64   // ceiling(25% of 255)
#define HALF_SPEED 128 // ceiling(50% of 255)
#define HIGH_SPEED 196 // ceiling(75% of 255)
#define FULL_SPEED 250 // ceiling(98% of 255)

#define CLOCKWISE_LETTER 'k'
#define ANTICLOCKWISE_LETTER 'j'
#define CLOCKWISE 1
#define ANTICLOCKWISE 0

#define MAX_MESSAGE_LENGTH 5

char commandKey;   // command to chose the motor & speed
char turn;      // 'k' turns clockwise, 'a' turns anticlockwise
static char command[MAX_MESSAGE_LENGTH];
static unsigned int command_pos = 1;

unsigned int sampleRate = 20U;
unsigned int window = 10;   // Number of samples taken

// LM35 (Temperature sensor)
float lmVoltage = 0.0;
```

```
float tempValue = 0.0;

// MPX (Pressure meter)
float mpxVoltage = 0.0;
float depthCoefficient =  1000 / (163.46 * 2.0);
int depthOffset = 10;
float depth = 0.0;

// TDS (TDS meter)
int analogBuffer[SCOUNT];    // store the analog value in the array, read from ADC
int analogBufferTemp[SCOUNT];
int analogBufferIndex = 0, copyIndex = 0;
int tdsValue = 0;
float tdsAverageVoltage = 0;

// TSW (Turbidity meter)
float tswVoltage = 0.0;
int ntuValue = 0;
```

Debug.hpp

```
#define DEBUG 1

#if DEBUG == 1
#define debug(x) Serial.print(x)
#define debugDec(x,d) Serial.print(x,d)
#define debugln(x) Serial.println(x)
#define debuglnDec(x,d) Serial.println(x,d)
#define debugDelay(x) delay(x)
#define debugRead() Serial.read()
#define debugAvailable() Serial.available()
#define debugMessages() verboseMessages()
#else
#define debug(x)
#define debugDec(x,d)
#define debugln(x)
#define debuglnDec(x,d)
#define debugDelay(x)
#define debugRead() 's'
#define debugAvailable() 1
#define debugMessages()
#endif
```

ScriptENTICMatlab.m

```
clear all;
delete(instrfind({'Port'},{'COM9'}));
s=serialport('COM9', 9600); %Port initialization
configureTerminator(s,"CR/LF")
fopen(s);
fid=fopen('resultats_Final.txt','w');
t=datetime('now'); pause(2);
i=0;
flushinput(s);

figure('Name','Measured Data','NumberTitle','off');
subplot(2,2,1)
h=animatedline;
ax= gca;

ax.YGrid = 'on';
ax.XGrid = 'on';
ax.YLim = [-10 10];
xlabel('TIME [s]');
ylabel('DEPTH [m]');
title('\bf{REAL TIME DEPTH}');

subplot(2,2,2)
h2=animatedline;
ay= gca;

ay.YGrid = 'on';
ay.XGrid = 'on';
ay.YLim = [20, 500];
xlabel('TIME [s]');
ylabel('TEMPERATURE [ºC]');
title('\bf{REAL TIME TEMPERATURE}');
```

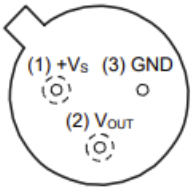
```
subplot(2,2,3)
h3=animatedline;
ax2= gca;

ax2.YGrid = 'on';
ax2.XGrid = 'on';
ax2.YLim = [100, 2000];
xlabel('TIME [s]');
ylabel('TDS [ppm]');
title('\bf{REAL TIME CONDUCTIVITY}');

subplot(2,2,4)
h4=animatedline;
ay2= gca;

ay2.YGrid = 'on';
ay2.XGrid = 'on';
ay2.YLim = [-500, 2000];
xlabel('TIME [s]');
ylabel('TURBIDITY [ntu]');
title('\bf{REAL TIME TURBIDITY}');

maxval=0;
minval=4;

variable=input('INGRESE EL VALOR DE LA VARIABLE: ', 's');

fwrite(s, variable);

% factor = (((5/1023)/163.46)/2)*1000;

if(variable=='S')
   startTime = datetime('now');
end
j=0;
while(variable=='S')

   lectura_bits=str2double(split(readline(s)));
   temp = lectura_bits(1);
   depth = lectura_bits(2);
   tds = lectura_bits(3);
   ntu = lectura_bits(4);

   %SERVEIX PER VEURE QUIN VALOR ÉS EL MÀXIM I QUIN EL MÍNIM
   if(depth>maxval)
      maxval=depth;
   if(depth<minval)
      minval=depth;
   end
   end

   t=datetime('now') - startTime;
   datenum(t);
   depth;
   addpoints(h, datenum(t), depth);
   addpoints(h2, datenum(t), temp);
   addpoints(h3, datenum(t), tds);
   addpoints(h4, datenum(t), ntu);
```

```
    %Updated axes
    ax.XLim = datenum([t-seconds(10) t]); datetick('x','keeplimits');
    ay.XLim = datenum([t-seconds(10) t]); datetick('x','keeplimits');
    ax2.XLim = datenum([t-seconds(10) t]); datetick('x','keeplimits');
    ay2.XLim = datenum([t-seconds(10) t]); datetick('x','keeplimits');
    drawnow
    pause(0.01);
    i=i+1;
    j=j+1;
end
fprintf(fid,'\n MAXIMUM DEPTH: %f m', maxval);
fprintf(fid, '\n MINIMUM DEPTH: %f m', minval);
fprintf(s,'%s\n', 'e');
fclose(fid);
```

DATASHEETS:

- **LM35**

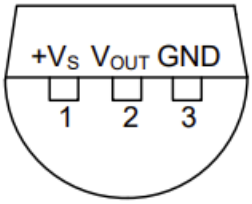
NDV Package
3-Pin TO-CAN
(Top View)



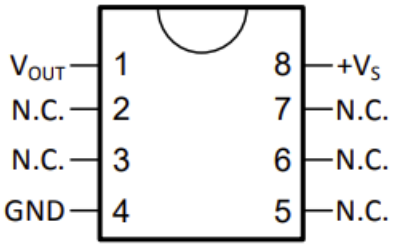
Case is connected to negative pin (GND)

Refer the second NDV0003H page for reference

LP Package
3-Pin TO-92
(Bottom View)

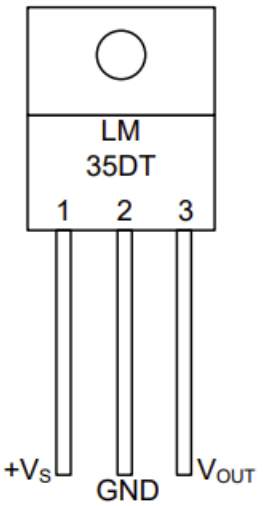


D Package
8-PIN SOIC
(Top View)



N.C. = No connection

NEB Package
3-Pin TO-220
(Top View)

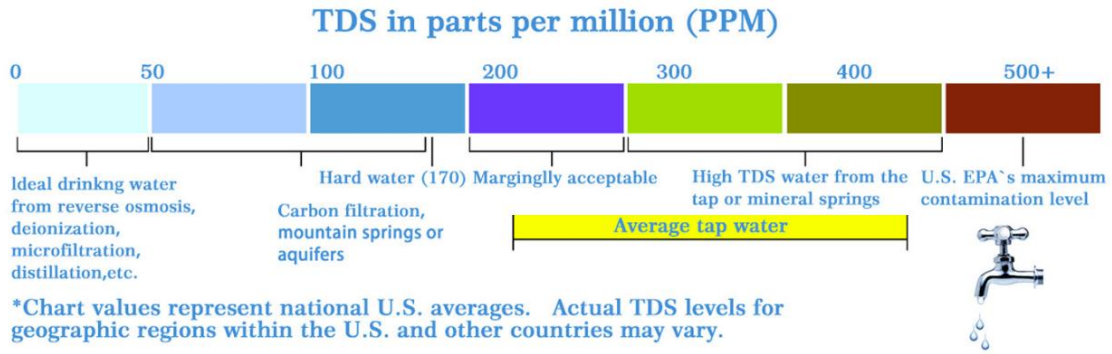


Tab is connected to the negative pin (GND).

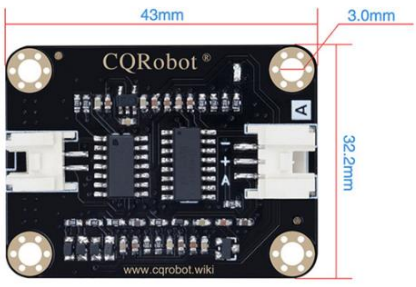
NOTE: The LM35DT pinout is different than the discontinued LM35DP

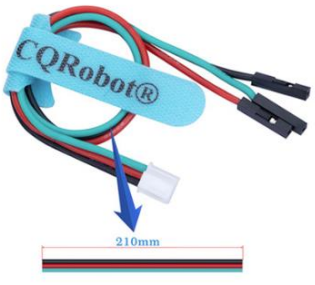
- **TDS (Total Dissolved Solids) Meter Sensor SKU: CQRSENTDS01**

Description



Size display





Specifications

Signal Transmitter Board Specifications

- Input Voltage: 3.3V to 5.5V
- Output Voltage: 0 to 2.3V
- Working Current: 3mA to 6mA
- TDS Measurement Range: 0 to 1000ppm
- TDS Measurement Accuracy: Plus/Minus 10% F.S. (25 Degree Celsius)
- Module Size: 43mm * 32.2mm
- Module Interface: JST 2.0mm 3-Pin
- Electrode Interface: JST 2.54mm 2-Pin

TDS Probe Specifications

- Number of Needle: 2
- Total Length: 83cm

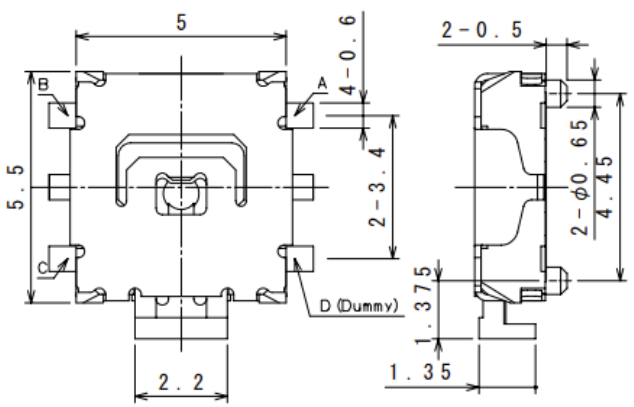
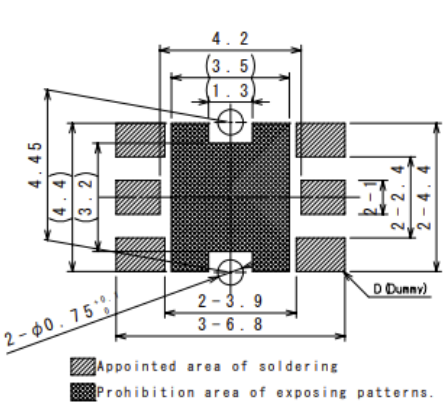
- **TSW-20-11B-1T40**

▯ Typical Specifications

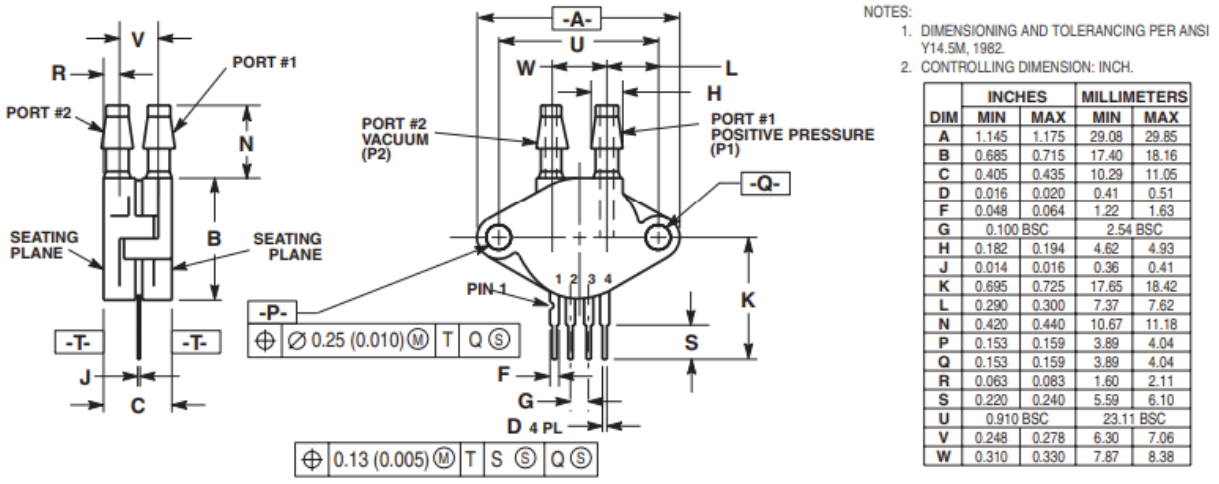
Item	Specifications
Ratings (max.) (Resistive load)	20mA 12VDC
Contact resistance	100 milliohm max. (Initial)
Insulation resistance	100 megohm min. 100V DC
Withstanding voltage	100V AC for 1min.
Operating life	100,000 cycles
Operating temperature range	-20 to +70 degree Celsius
Storage temperature range	-30 to +80 degree Celsius (except carrier tape)

▯ Dimensions

Unit : mm

No	Style	P.C.B reference Land Dimensions Circuit Diagram (TOP VIEW)
		

- **MPX2100AP**



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.

	INCHES		MILLIMETERS	
DIM	MIN	MAX	MIN	MAX
A	1.145	1.175	29.08	29.85
B	0.685	0.715	17.40	18.16
C	0.405	0.435	10.29	11.05
D	0.016	0.020	0.41	0.51
F	0.048	0.064	1.22	1.63
G	0.100 BSC		2.54 BSC	
H	0.182	0.194	4.62	4.93
J	0.014	0.016	0.36	0.41
K	0.695	0.725	17.65	18.42
L	0.290	0.300	7.37	7.62
N	0.420	0.440	10.67	11.18
P	0.153	0.159	3.89	4.04
Q	0.153	0.159	3.89	4.04
R	0.063	0.083	1.60	2.11
S	0.220	0.240	5.59	6.10
U	0.910 BSC		23.11 BSC	
V	0.248	0.278	6.30	7.06
W	0.310	0.330	7.87	8.38

STYLE 1:
PIN 1. GROUND
2. + OUTPUT
3. + SUPPLY
4. - OUTPUT

CASE 344C-01
ISSUE B
UNIBODY PACKAGE

Figure 3 shows the output characteristics of the MPX2100 series at 25°C. The output is directly proportional to the differential pressure and is essentially a straight line.

The effects of temperature on Full-Scale Span and Offset are very small and are shown under Operating Characteristics.

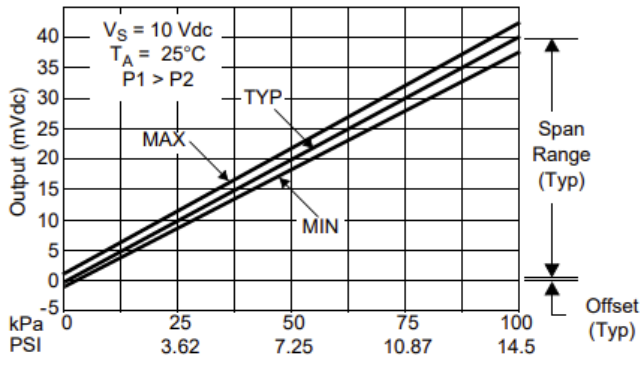


Figure 1 shows a block diagram of the internal circuitry on the stand-alone pressure sensor chip.

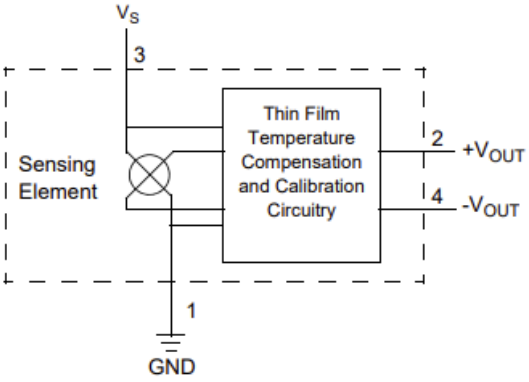
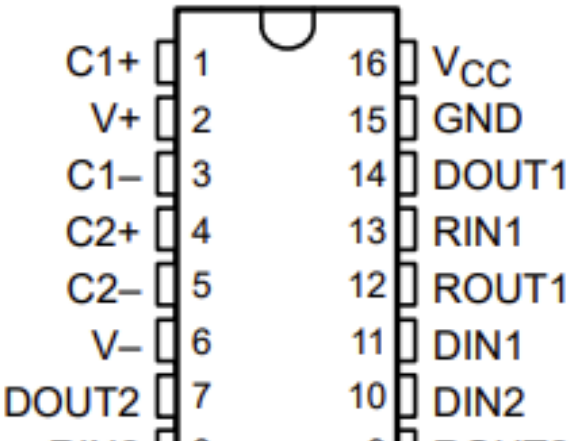


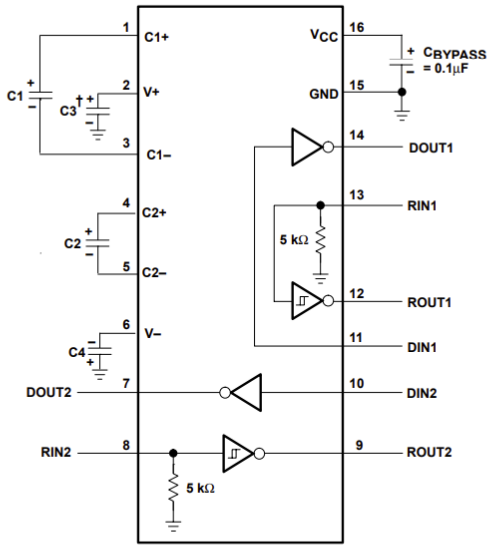
Figure 1. Temperature Compensated Pressure Sensor Schematic

- **MAX3232**

D, DB, DW, OR PW PACKAGE
(TOP VIEW)



APPLICATION INFORMATION



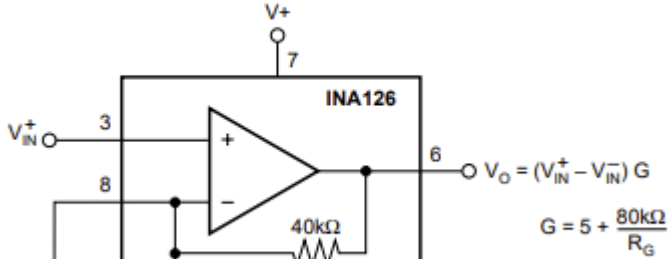
† C3 can be connected to V_{CC} or GND.

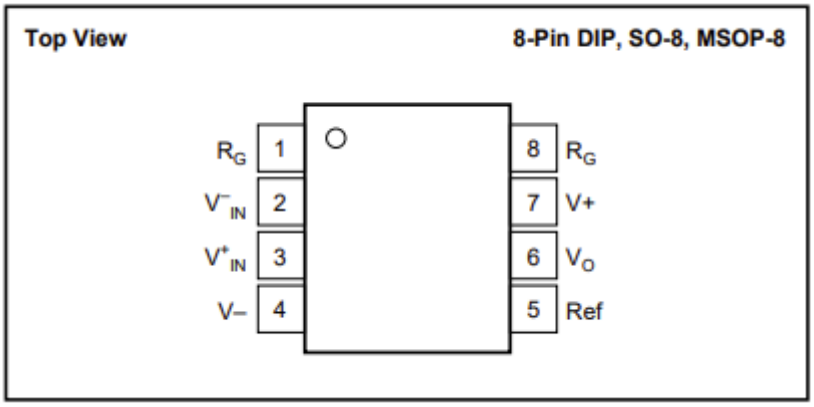
V_{CC} vs CAPACITOR VALUES

V_{CC}	C1	C2, C3, C4
3.3 V ± 0.3 V	0.1 µF	0.1 µF
5 V ± 0.5 V	0.047 µF	0.33 µF
3 V to 5.5 V	0.1 µF	0.47 µF

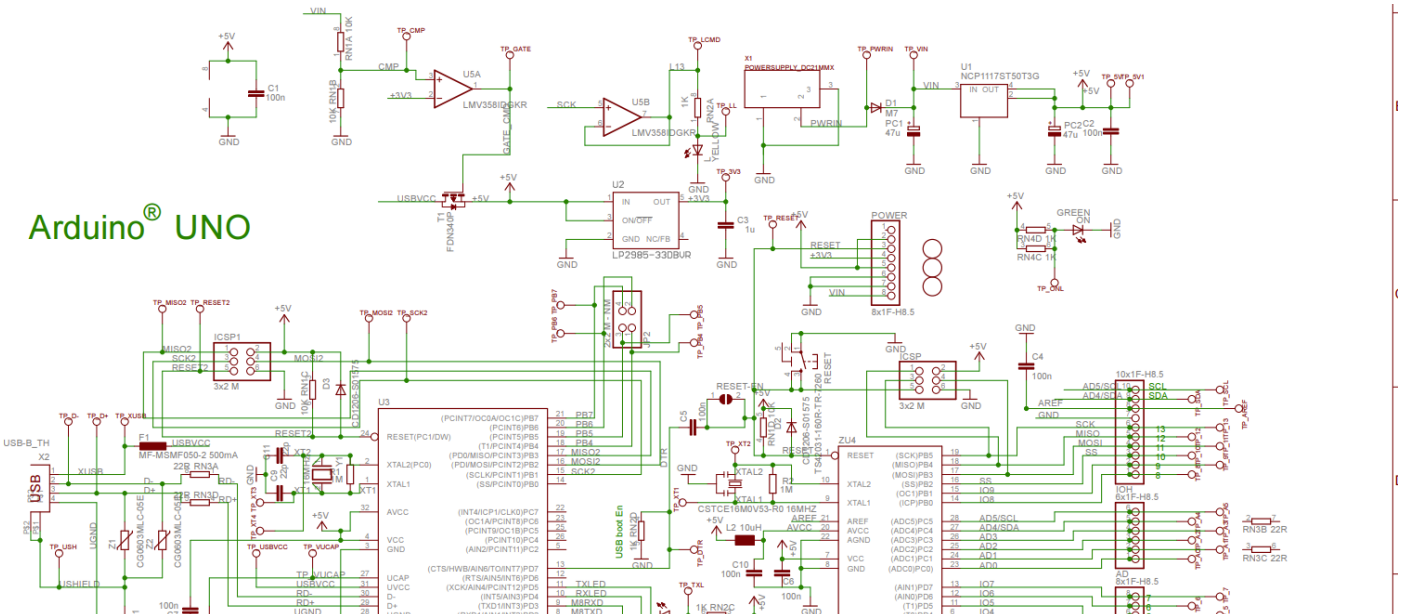
Figure 4. Typical Operating Circuit and Capacitor Values

- **INA126**




Top View
8-Pin DIP, SO-8, MSOP-8
RG 1
V−IN 2
V+IN 3
V− 4
8 RG
7 V+
6 VO
5 Ref

- **ARDUINO UNO**


Arduino® UNO

- **DRV8876**

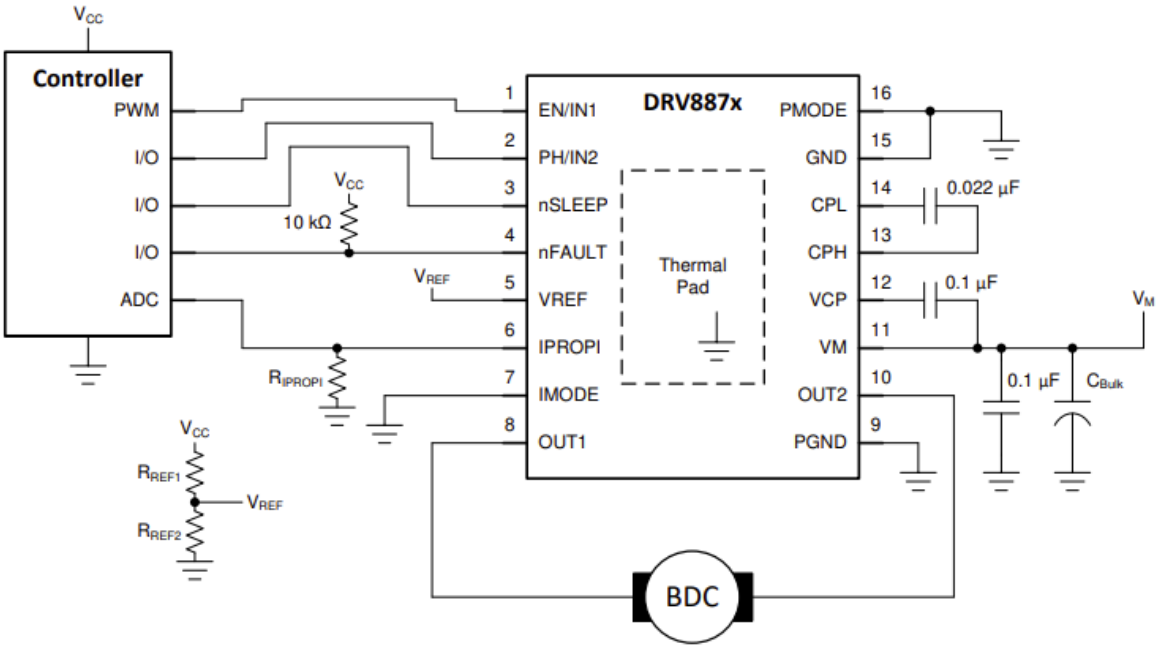


Figure 18. Typical Application Schematic

- **DB9**

9-PIN	Nombre		Sentido	Función
5	GND	Ground	-----	Conductor conectado al chasis.
3	TXD	Datos de Transmisión	DTE -> DCE	Datos de usuario.
2	RXD	Datos de Recepción	DTE <- DCE	Datos de usuario.
7	RTS	Request To Send	DTE -> DCE	DTE tiene datos para transmitir.
8	CTS	Clear To Send	DTE <- DCE	Permiso al DTE para transmitir.

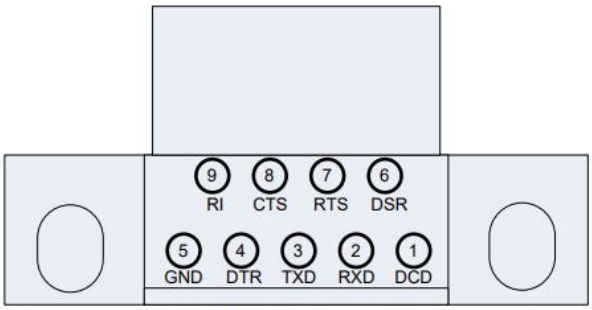


Figure 2.1 – DB9-USB-RS232-M Pin-Out from a Top View through the module

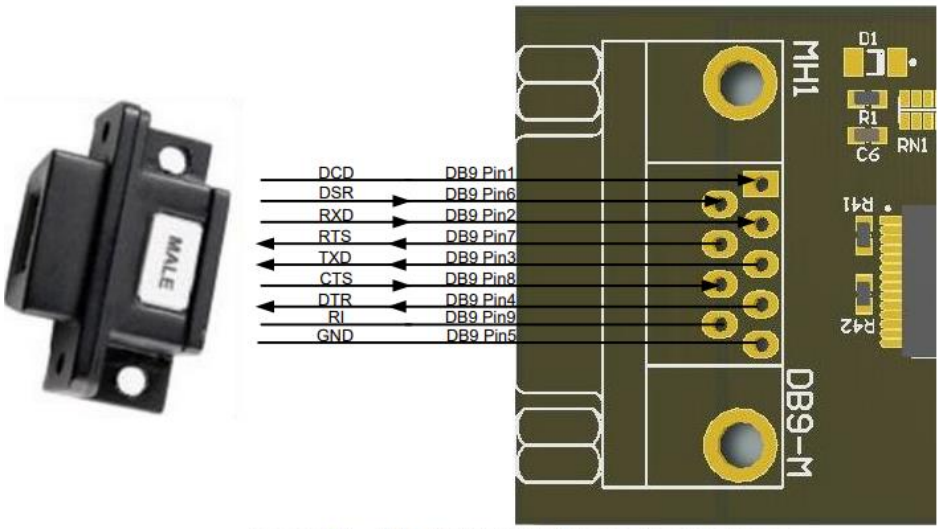


Figure 2.2 – DB9-USB-RS232-M Connection illustration