Académie Aix-Marseille Lycée Alphonse Benoît SESSION 2022

RAPPORT DE PROJET
REVUE FINALE

HighLine / SlackLine

BENNOUR RUFFIN MERCHAT FLEURY SPINA RICARD DOYER SOUALMIA

Sommaire

Partie Commune

Contexte :

L'entreprise ROPE & WEB nous a contactés pour pouvoir les aider dans le développement d'un système qui permet la sécurisation de différents systèmes de secours sur corde qui peuvent être délicats à installer et qui nécessitent une grande précision pour éviter tout sur-accident.
Exemple d'installation :

Cette entreprise est aussi chargée de l'installation et de la sécurité de la slackline (une corde tendue entre deux points) pour que Nathan PAULIN puisse exercer sa discipline highline en toute sécurité.
Il est question en 2024 pour les jeux olympiques de tendre une slackline entre la Tour Eiffel et la Tour Montparnasse, soit une distance de 2700 m

Nous, le BTS SN du lycée BENOIT sommes en charge d'équiper cette slackline de capteurs afin d'aider le physicien de la société à modéliser la corde pendant le trajet de Nathan Paulin, de mettre en évidences les forces en jeu, de pouvoir prévenir d'éventuel dangers grâce à différentes alarmes en cas de dépassement de seuil définis en amont par l'entreprise.

Rôle de chaque étudiants :

IR :
Equipe 1 :

BENNOUR Mohamed-Rayan **IR 1.1**	Liste des tâches assurées par l'étudiant : -Cas d'utilisation 'Acquérir'	**Installation :** EDI qt-creator **Mise en œuvre :** Programmation C++/Qt, communication I2C, USB, Bluetooth, série **Configuration :** EDI, GPIO, Bluetooth. **Réalisation :** Codage de l'acquisition des valeurs des capteurs angulaires, dynamomètre, anémomètre, girouette. **Documentation** : Manuel d'installation.
RUFFIN Thomas **IR 1.2**	Liste des tâches assurées par l'étudiant -Cas d'utilisation 'Superviser', 'Sauver dans le cloud', 'Régler seuils alarme', 'alarmer', 'Afficher'.	**Installation :** EDI QtCreator **Mise en œuvre :** Programmation C++/Qt, EDI qt-creator, Fichier csv, communications réseau **Configuration :** WIFI, EDI, réseau, cloud, firewall **Réalisation :** Codage du programme principal et de l'IHM avec les fonctionnalités liées aux cas d'utilisation attribués. **Documentation :** Manuel de pannes.

EC :
Equipe 1 :

FLEURY Benjamin **EC 1.3**	Liste des tâches assurées par l'étudiant **Conception d'un capteur angulaire / Rapporteur numérique communiquant** -La mesure se fera à partir d'un potentiomètre muni de 2 bras, associé à un ESP8266 ou à un ESP32S3 Lilygo. -Prévoir une alarme sonore en cas de dépassement de seuil. -L'affichage de la mesure sera local (afficheur), mais cette mesure pourra aussi être transmise vers un RPI par Wi-Fi, et par bus I2C. -L'alimentation de l'ensemble se fera par bloc secteur. -Produire le schéma structurel de l'ensemble. -Effectuer le routage d'une carte fille et produire les fichiers afin que la fabrication du PCB soit sous-traitée. -Câbler la carte et effectuer les essais. -Documenter la mise en service de la carte finalisée.	**Installation :** Mise en service (init./config.) : - d'un Raspberry Pi (librairie BCM2835, Qt Creator, autres si nécessaire) - d'un IDE pour ESP8266 / ESP32 S3 **Mise en œuvre :** Concevoir une structure de mesure d'angle entre 2 cordes/câbles amarrées en un même point d'ancrage. Des essais se feront dans un premier temps par câblage rapide sur un rapporteur numérique associé à une Arduino Uno. Puis seront transposés sur un outil disposant d'un afficheur. La carte offrira la possibilité à l'utilisateur de transmettre cette mesure au RPI par Wi-Fi, et par bus I2C. Une fonctionnalité (option avancée du contrat) sera la possibilité de mettre la carte en esclave Modbus RTU RS485 pour la transmission des mesures. Une alarme sonore en cas de dépassement de seuils réglables sera prévue. Proposer un schéma structurel de l'ensemble. **Réalisation :** Après validation de la solution, concevoir un circuit imprimé devant être fabriqué industriellement. **Documentation :** -Schéma de câblage rapide (Fritzing) pour documenter la phase d'essais. -Documents de fabrication de la carte (KiCAD). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. -Schéma structurel avec contours IBD. Liste complète des composants avec leur source d'approvisionnement, code commande et prix. Programmes en C/C++, accompagnés des commentaires et diagrammes nécessaires à sa compréhension. -Fiche de mise en service. Fiche de dépannage.

SPINA Rémi **EC 1.4**	Liste des tâches assurées par l'étudiant **Mesure de force (poids) sur mâts de déport** -La mesure se fera à partir de jauges de contraintes, associées à un convertisseur spécialisé, et géré par un ESP8266 ou à un ESP32S3 Lilygo. -Prévoir une alarme sonore en cas de dépassement de seuil réglable. -L'affichage de la mesure sera local (afficheur), mais cette mesure pourra aussi être transmise vers un RPI par Wi-Fi, et par bus I2C. -L'alimentation de l'ensemble se fera par bloc secteur. -Produire le schéma structurel de l'ensemble. -Effectuer le routage d'une carte fille et produire les fichiers afin que la fabrication du PCB soit sous-traitée. -Câbler la carte et effectuer les essais. Documenter la mise en service de la carte finalisée.	Installation : Mise en service (init./config.) : - d'un Raspberry Pi (librairie BCM2835, Qt Creator, autres si nécessaire) - d'un IDE pour ESP8266 / ESP32 S3 **Mise en œuvre** : Concevoir une structure de mesure de charge pour mâts de déport. Des essais sur l'association jauges de contraintes / convertisseur HX711 se feront dans un premier temps sur breakout avec Arduino Uno. Puis seront transposés sur un outil disposant d'un afficheur. La carte offrira la possibilité à l'utilisateur de transmettre cette mesure au RPI par Wi-Fi, et par bus I2C. Une fonctionnalité (option avancée du contrat) sera la possibilité de mettre la carte en esclave Modbus RTU RS485 pour la transmission des mesures. Une alarme sonore en cas de dépassement de seuils réglables sera prévue. Proposer un schéma structurel de l'ensemble. **Réalisation :** Après validation de la solution, concevoir un circuit imprimé devant être fabriqué industriellement. **Documentation :** -Schémas de câblage rapide (Fritzing) pour documenter la phase d'essais. -Documents de fabrication de la carte (KiCAD). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. -Schéma structurel avec contours IBD. Liste complète des composants avec leur source d'approvisionnement, code commande et prix. -Programmes en C/C++ , accompagnés des commentaires et diagrammes nécessaires à sa compréhension. -Fiche de mise en service. -Fiche de dépannage.

Tous les étudiants	Tâches à traiter par l'ensemble des étudiants de l'équipe projet pour le développement de la solution Documents de vie du projet : - Fiches de lecture croisée - Comptes rendus de réunion. Domaines de physique à traiter par l'ensemble des étudiants de l'équipe projet : Les capteurs, les antennes, les OEM, modulations numériques, WIFI. Puissance et énergie. Filtrage numérique	Intégration de la solution et livraison au client du matériel/logiciel/sources/manuels. Le logiciel sera installable facilement chez le client en suivant une procédure écrite.

Annexe *:*

Diagramme exigences :

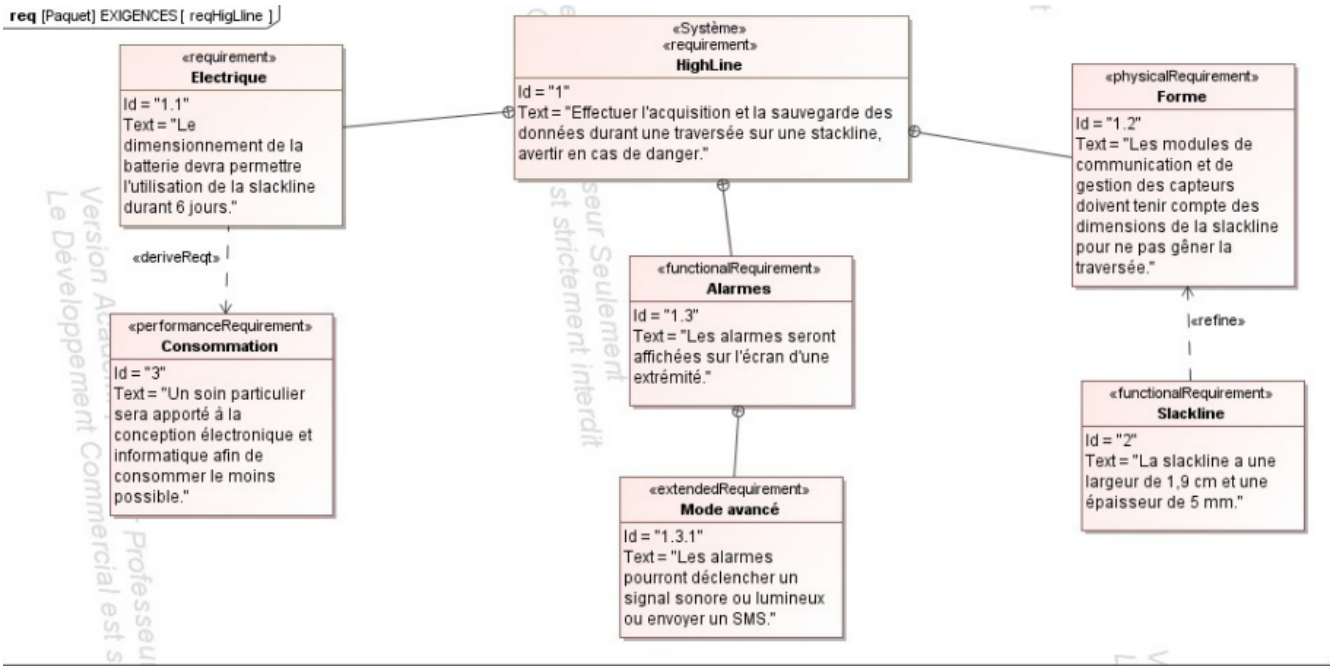


diagramme de déploiement du système :

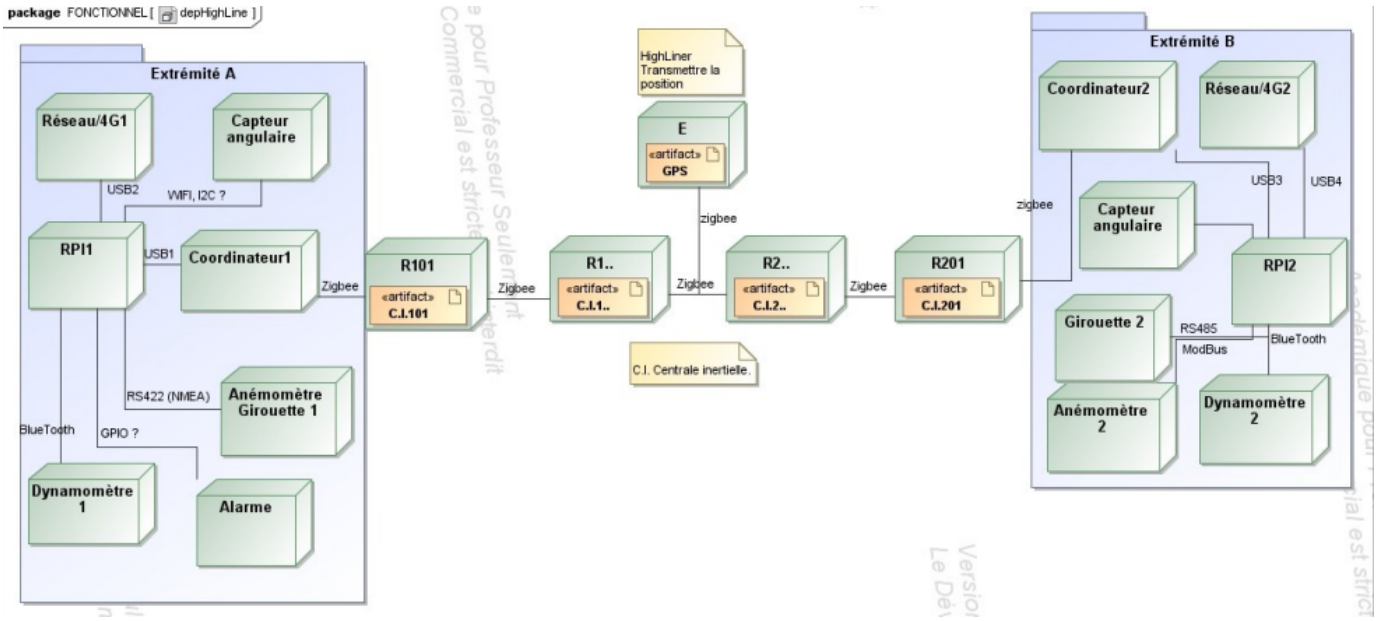


Diagramme de cas d'utilisations :

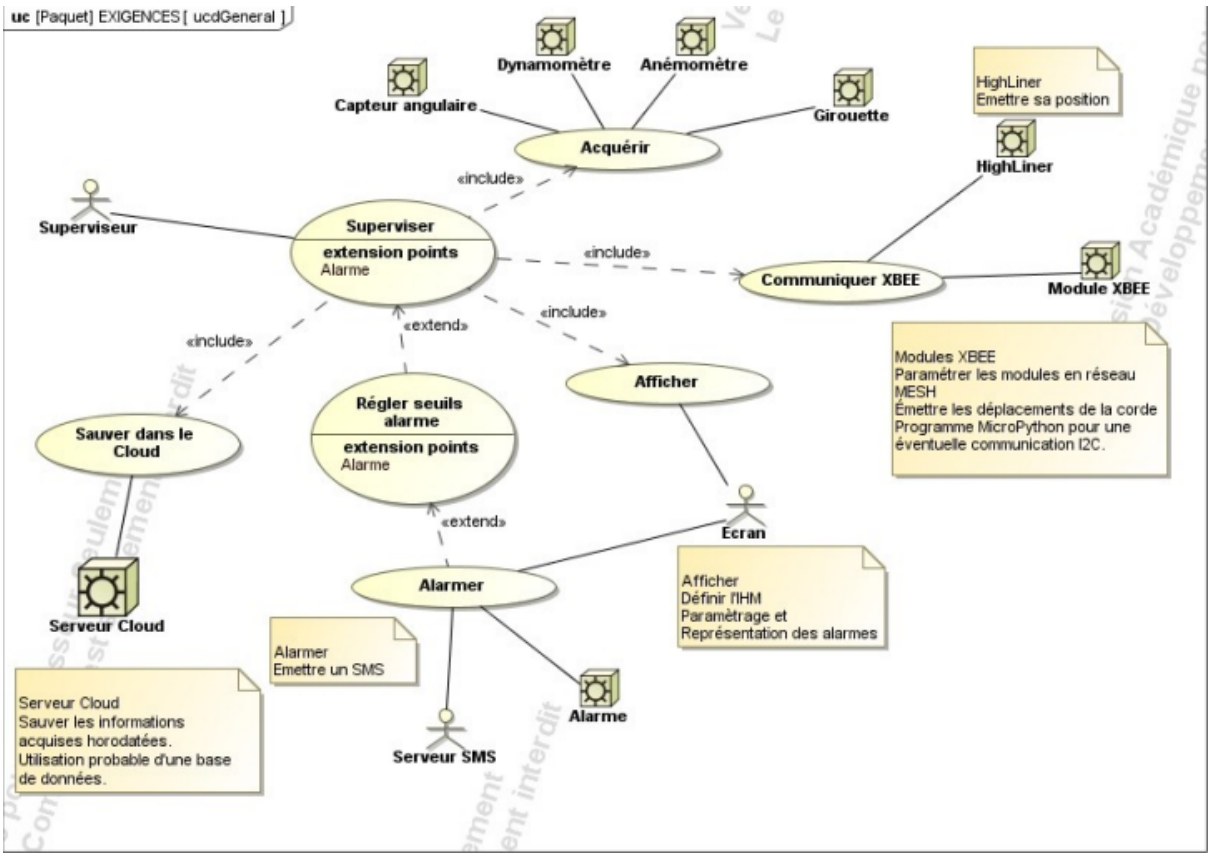
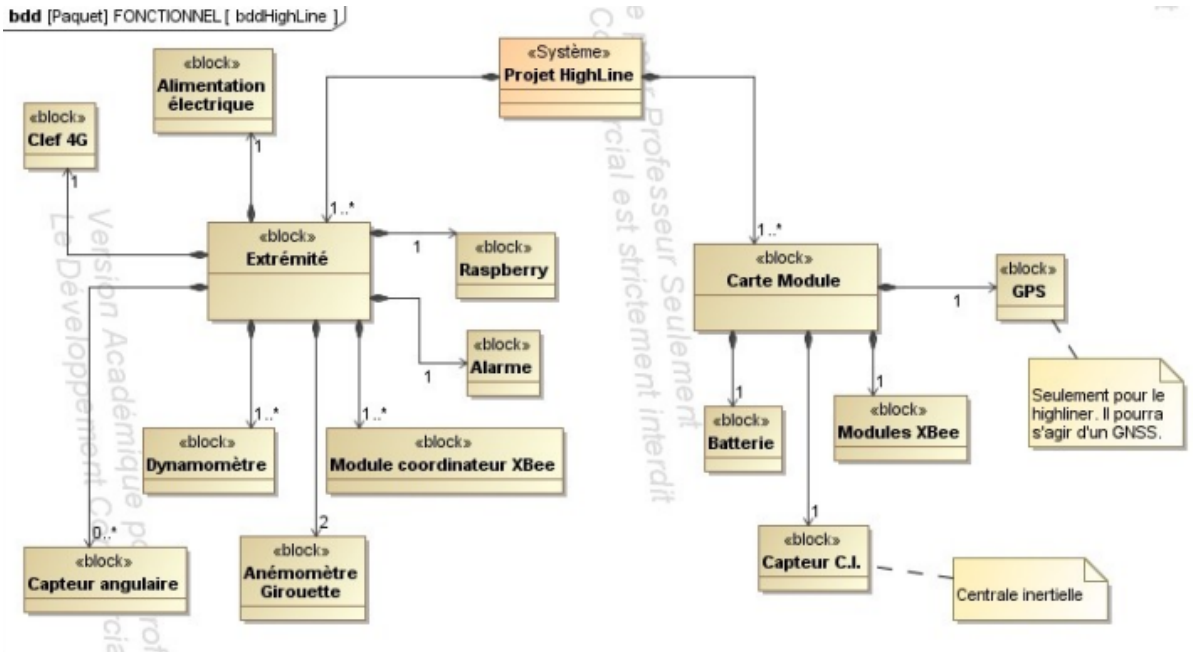


Diagramme de blocs :



Partie individuelle
Bennour Mohamed-Rayan

Mon rôle dans ce projet est d'effectuer différents programmes sous Qt Creator, qui permettront d'acquérir et décoder les valeurs que renvoient une girouette, un anémomètre, un dynamomètre, les différents capteurs angulaires, ainsi qu'un appareil qui servira comme mélange entre girouette et anémomètre, qui n'a pas encore été conçu.

Diagramme de Gantt prévisionnelle:

1			**Projet**	**221 h**	**Mar 03/01/2**	**Mar 06/06/2**		
2			**Tâches communes**	**10 h**	**Mar 03/01/2**	**Mer 04/01/2**		
3			Lecture du dossier	3 h	Mar 03/01/2:	Mar 03/01/2:		Maxence;MR;Thom
4			Réunion de début de projet	1 h	Mar 03/01/2:	Mar 03/01/2:	3	Maxence;MR;Thom
5			Compréhension du contrat, découverte matériel et documentation	6 h	Mer 04/01/23	Mer 04/01/23	4	MR;Maxence; Thomas
6			Planification prévisionnelle	2 h	Mar 10/01/2:	Mar 10/01/2:	5	MR
7			Se renseigner sur la capacité à communiquer du dynamomètre	2 h	Mar 10/01/23	Mar 10/01/23	6	MR[30%]
8			Spécification concernant le capteur Anémomètre et Girouette	7 h	Mar 10/01/23	Mer 11/01/23	6	MR[70%]
9			Programme de test du modbus	14 h	Mer 11/01/2	Mer 18/01/2	8	MR
10			Programme de lecture de la vitesse et direction du vent	20 h	Lun 23/01/23	Mar 31/01/23	9	MR
11			Rédaction 1 du rapport de projet	4 h	Mer 01/02/2	Mer 01/02/2	10	MR
12			Codage CAnemometre2 et CGirouette2 et intégration	7 h	Mer 01/02/23	Mar 07/02/23	11	MR
13			REVUE 1	30 min	Mar 07/02/2:	Mar 07/02/2:		

Diagramme de Gantt final:

1	**Projet**	**221 h**	**Mar 03/01/2**	**Mar 06/06/2**		
2	**Tâches communes**	**10 h**	**Mar 03/01/2**	**Mer 04/01/2**		
3	Lecture du dossier	3 h	Mar 03/01/2:	Mar 03/01/2:		Maxence;MR;Thom
4	Réunion de début de projet	1 h	Mar 03/01/2:	Mar 03/01/2:	3	Maxence;MR;Thom
5	Compréhension du contrat, découverte matériel et documentation	6 h	Mer 04/01/23	Mer 04/01/23	4	MR;Maxence; Thomas
6	Planification prévisionnelle	2 h	Mar 10/01/2:	Mar 10/01/2:	5	MR
7	Se renseigner sur la capacité à communiquer du dynamomètre	2 h	Mar 10/01/23	Mar 10/01/23	6	MR[30%]
8	Spécification concernant le capteur Anémomètre et Girouette	7 h	Mar 10/01/23	Mer 11/01/23	6	MR[70%]
9	Programme de test du modbus	7 h	Mer 11/01/2	Mar 17/01/2:	8	MR
10	Programme de lecture de la direction du vent	27 h	Mar 17/01/23	Mar 31/01/23	9	MR
11	Rédaction 1 du rapport de projet	4 h	Mer 01/02/2	Mer 01/02/2	10	MR
12	Codage CGirouette2 et intégration	7 h	Mer 01/02/2	Mar 07/02/2:	11	MR

J'ai dû retirer l'élaboration du programme de l'anémomètre par manque de temps.

Renseignements sur le dynamomètre:

Après m'être familiarisé avec les différents instruments, un problème se révéla très vite: on ne peut pas communiquer avec le dynamomètre fourni. Effectivement, l'entreprise ayant créé le produit, LineGrip, ne mentionne pas le moyen de communication utilisé. Cela n'a sûrement pas été jugé nécessaire, vu que le dynamomètre, le linescale 3, a une interface directement sur l'appareil. Malgré tout, il est indiqué sur le site web de LineGrip qu'il est possible de connecter le dynamomètre à un téléphone via bluetooth. Et qu'ensuite, grâce à une application fournie, de visualiser les informations voulues.

Malheureusement, sans ces informations, il est impossible de récupérer les données pour les exploiter dans le programme principal.

Ma première tâche à donc été, étonnamment, d'écrire un mail au contact de l'entreprise Rope&Web, Quentin Camus. Selon mes professeurs, il posséderait des contacts chez LineGrip.

Voici le mail en question:

Bonjour Monsieur Camus,

Je suis Mohamed-Rayan BENNOUR, un étudiant de BTS en Systèmes Numériques du lycée Alphonse Benoit, à L'Isle sur la Sorgue.

Pour mon projet de fin d'année, monsieur Antoine m'a confié la mission d'effectuer l'acquisition des données qu'envoient différents capteurs, pour le projet de la slackline.

Malheureusement, il nous manque des informations cruciales afin de développer un programme permettant de communiquer avec le dynamomètre lineGrip lineScale 3 que vous utilisez, tel que le protocole utilisé, et par quel moyen de communication.

Selon monsieur Antoine, vous disposez d'un contact en langue française chez LineGrip, qui pourrait être en mesure de répondre à nos questions.

Vous remerciant par avance des renseignements que vous pourrez nous fournir,

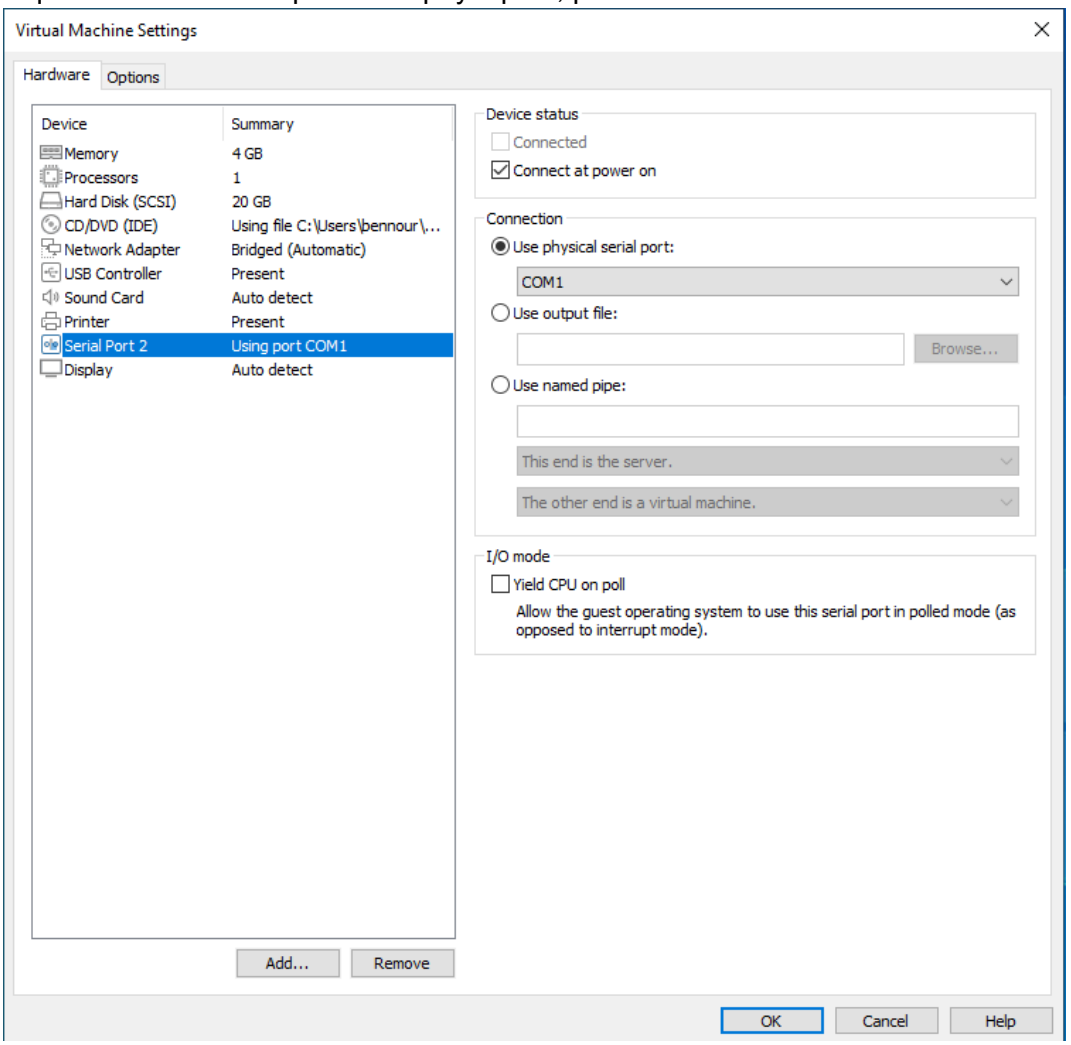
je vous prie d'agréer, Monsieur, mes salutations les plus distinguées.

Une fois sa réponse obtenue, je referai donc un autre mail, cette fois au contact de chez LineGrip.

Préparation communication girouette:

Mais pour le moment, il est impossible de travailler sur le dynamomètre, je commence donc par la girouette. Vu qu'on travaillait en même temps sur la girouette moi et l'étudiant RICARD Loïc, je cherche donc un autre moyen de tester le programme que j'allais concevoir. Mes professeurs me recommandent de communiquer entre mon ordinateur, et la machine virtuelle Linux que j'y avais installé.

Pour ce faire, il faut d'abord connecter l'ordinateur à lui-même. On branche un adaptateur USB vers série sur le port qui sera attribué à la machine virtuelle. Ensuite on connecte un autre câble série à la sortie de l'adaptateur, et on le connecte au port série. Ceci fait, ce n'est pas encore terminé vu qu'il faut encore paramétrer la machine virtuelle. Il faut donc aller dans les paramètres de celle-ci, puis dans l'onglet des paramètres port série. Il faut ensuite cliquer sur «utiliser un port série physique», puis choisir COM1.



Après cela, il y a encore une étape. Afin que la machine virtuelle Linux puisse lire et écrire à partir d'un périphérique, il faut ajouter l'utilisateur au groupe «dialout».
On doit donc exécuter la commande suivante: sudo usermod -aG dialout «utilisateur».

Je peux donc maintenant communiquer entre mon pc et ma machine virtuelle avec un programme.

Programme test communication:

D'abord le .h :

```h
#ifndef MAINUI_H
#define MAINUI_H

#include <QDialog>
#include <QSerialPort>

QT_BEGIN_NAMESPACE
namespace Ui { class MainUI; }
QT_END_NAMESPACE

class MainUI : public QDialog
{
    Q_OBJECT

public:
    MainUI(QWidget *parent = nullptr);
    ~MainUI();

private slots:
    void on_pushButton_clicked();

private:
    Ui::MainUI *ui;
    QSerialPort * _com;
};
#endif // MAINUI_H

```

Puis le .cpp :

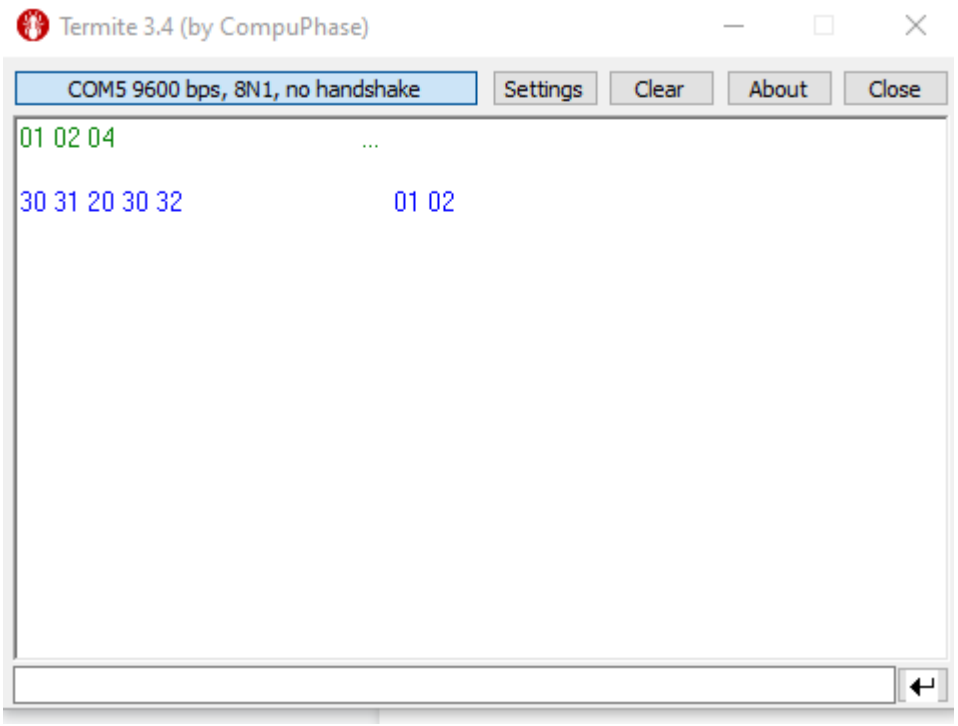
```cpp
#include "mainui.h"
#include "ui_mainui.h"
#include <QDebug>
#include <QSerialPortInfo>

MainUI::MainUI(QWidget *parent)
    : QDialog(parent)
    , ui(new Ui::MainUI)
{
    ui->setupUi(this);

    QList<QSerialPortInfo> listePorts = QSerialPortInfo::availablePorts();

    for(QSerialPortInfo port : listePorts) {
        ui->cbxPortsSerie->addItem(port.portName());
    }
}

MainUI::~MainUI()
{
    delete ui;
}

void MainUI::on_pushButton_clicked()
{
    QString tty = ui->cbxPortsSerie->currentText();

    _com = new QSerialPort(tty);

```

```
30      // Configure le port
31      _com->setBaudRate(QSerialPort::Baud9600);
32      _com->setDataBits(QSerialPort::Data8);
33      _com->setParity(QSerialPort::NoParity);
34      _com->setStopBits(QSerialPort::OneStop);
35
36      // Ouvrir le port
37      bool isOk = _com->open(QIODevice::ReadWrite);
38
39      //
40      if(isOk) {
41          QByteArray trame("\x01\x02\x04");
42
43          _com->write(trame.data());
44      } else {
45          qDebug() << "Pb ouverture port serie";
46      }
47          if (_com->waitForReadyRead(5000)) {
48              QByteArray data = _com->readAll();
49              qDebug() << "Recu:" << data;
50          } else {
51
52
53              qDebug() << "Données non conforme";
54          }
55  }
```

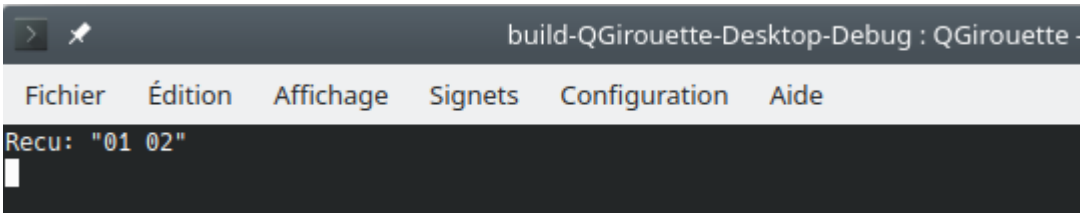
Ce programme, qui permet de se connecter au port série COM1 (ttyUSB0), envoie les octets suivants: 0x01, 0x02, 0x04. Puis permet de lire une trame que l'on renvoie.

A l'aide d'un logiciel permettant de recevoir ou d'envoyer des trames, tel que commix ou termite, on vérifie si la trame est bien reçue par l'ordinateur physique.

On doit dans les réglages, utiliser le véritable port série (COM5). Lorsque l'on exécute le programme sur Qt Creator, on voit bien apparaître notre trame sur termite.



On renvoie 01 02 à notre programme, et le qDebug nous affiche notre résultat dans la console.



Protocole modbus

Avant d'améliorer le programme, il est important de connaître le protocole employé par la girouette, le protocole modbus.

J'ai donc passé un long moment à me renseigner sur celui-ci. C'est un protocole maître-esclave, ou l'esclave à l'autorisation d'envoyer des données uniquement lorsque le maître lui demande. Pour ce faire, le maître envoie une trame composée de 8 octets.

Le premier octet est l'adresse de l'appareil esclave : l'adresse de l'appareil esclave est utilisée pour identifier l'appareil avec lequel le maître veut communiquer.

Le deuxième octet est le code de fonction : le code de fonction détermine l'action que le maître veut effectuer sur l'appareil esclave. Il peut s'agir de la lecture ou de l'écriture de données, de l'exécution de commandes, etc.

Ensuite, on a sur 2 octets l'adresse de départ : ils spécifient l'adresse du premier registre de données (emplacement mémoire dans l'esclave) que le maître veut lire ou écrire.

Les 2 octets suivants correspondent au nombre d'éléments (2 octets) : ils spécifient le nombre de registres de données que le maître veut lire ou écrire.

Les 2 derniers octets correspondent au CRC : le CRC (Cyclic Redundancy Check) est un code de contrôle de redondance cyclique qui permet de vérifier l'intégrité des données de la trame.

La trame que renverra l'esclave se composera de 7 octets. Les 2 premiers octets sont identiques à ceux du maître, c'est à dire l'adresse de l'esclave est l'action à effectuer. Les 3 prochains octets correspondent aux données. Les octets que l'on veut exploiter sont les octets 4 et 5. Les 2 derniers octets correspondent au CRC.

Voici comment se compose une trame modbus:

	Adresse esclave	Code Requête	Données	contrôle	
START	Adresse	Fonction	Données	CRC	END
Silence	1 octet	1 octet	n octets	2 octets	Silence

Réalisation du programme de la girouette :

Maintenant que je maîtrise le modbus, je peux faire le programme qui permet d'acquérir les données de la girouette dans son entièreté.

Voici le programme final, ou la classe CGui est composée de la classe CModBusRTU.

D'abord, voici le fichier cmodbusrtu.h :

```
#ifndef CMODBUSRTU_H
#define CMODBUSRTU_H

#include <QObject>
#include <QByteArray>

class CModBusRTU : public QObject
{
    Q_OBJECT
public:
    explicit CModBusRTU(QObject *parent = nullptr);
    bool comparerCrC(uint16_t crcFourni);
    QByteArray formerRequete(uint8_t noEscl, uint8_t codeFonc,
                             uint16_t adrBase, uint16_t nbReg);
    void setTrame(QByteArray data);
    void getTrame(uint8_t *trame);
    QByteArray extraireDatasReponse();

private:
    QByteArray _trame;
    uint8_t _datas[255];
    uint8_t _nb;
    uint16_t calculerCrc16(int nb);

signals:

};

#endif // CMODBUSRTU_H

```

Puis le fichier cgui.h :

```
#ifndef CGUI_H
#define CGUI_H

#include <QMainWindow>
#include <QtSerialPort>
#include "cmodbusrtu.h"

QT_BEGIN_NAMESPACE
namespace Ui { class CGui; }
QT_END_NAMESPACE

class CGui : public QMainWindow
{
    Q_OBJECT

public:
    CGui(QWidget *parent = nullptr);
    ~CGui();

private slots:
    void on_pbConnecter_clicked();
    void on_pbEnvoyer_clicked();

private:
    Ui::CGui *ui;
    QSerialPort * _com;
    CModBusRTU _mb;
    QByteArray _reponse;
    QByteArray _data;
    uint8_t _trame[255];
    uint16_t _crc;

};
#endif // CGUI_H

```

Des slots sont initialisés car j'ai créé une interface qt pour tester le programme.

Maintenant voici le fichier cmodbusrtu.cpp :

```
45      if (crcCalc != crcFourni) return false;
46      else return true;
47  }
48
49  uint16_t CModBusRTU::calculerCrc16(int nb)
50  {
51      uint8_t nbDec;
52      uint8_t yAUn;
53      uint8_t indice;
54      uint16_t crc;
55
56      crc = 0xFFFF;
57      indice = 0;
58      do {
59        crc ^= _trame[indice];
60        nbDec=0;
61        do {
62            if (crc&0x0001) yAUn=1;
63            else yAUn=0;
64            crc >>= 1;
65            if (yAUn)
66                crc ^= 0xA001;
67            nbDec++;
68        } while(nbDec < 8);
69        indice++;
70      } while (indice < nb);
71      qDebug() << "crc calcule" << crc;
72      return crc;
73  }
74
75  QByteArray CModBusRTU::extraireDatasReponse() {
76      QByteArray qba;
77
78      qba.append(_trame.at(3));
79      qba.append(_trame.at(4));
80      return qba;
81  }
82
```

La totalité des méthodes de la classe CModBusRTU m'ont été fournies, vu que le niveau de complexité de leurs programmation dépasse clairement mon niveau.

J'ai passé plusieurs heures à essayer de comprendre la totalité de ces méthodes, afin de pouvoir les utiliser correctement. J'ai d'ailleurs eu énormément de mal à comprendre le fonctionnement de la fonction memcpy dans la méthode getTrame, ou encore pourquoi on emploie des "static_cast" dans la méthode formerRequete. La méthode calculerCrc16 m'est d'ailleurs incompréhensible vu qu'elle emploie des formules mathématiques très complexes, mais je sais qu'elle renvoie la valeur du crc d'une trame, et que sa valeur est infaillible.

Voici maintenant le fichier cgui.cpp:

```cpp
#include "cgui.h"
#include "ui_cgui.h"
#include <QDebug>
#include <QtSerialPort>

CGui::CGui(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::CGui)
{
    ui->setupUi(this);

    QList<QSerialPortInfo> listePorts = QSerialPortInfo::availablePorts();

    for(QSerialPortInfo port : listePorts) {
        ui->cbxPortsSerie->addItem(port.portName());
    }

}


CGui::~CGui()
{
    delete _com;
    delete ui;
}


void CGui::on_pbConnecter_clicked()
{
    QString tty = "/dev/"+ui->cbxPortsSerie->currentText();
    _com = new QSerialPort(tty);
    // Configure le port
    _com->setBaudRate(QSerialPort::Baud9600);
    _com->setDataBits(QSerialPort::Data8);
    _com->setParity(QSerialPort::NoParity);
    _com->setStopBits(QSerialPort::OneStop);
    _com->setFlowControl(QSerialPort::NoFlowControl);
    // Ouvrir le port
    _com->open(QIODevice::ReadWrite);
}

```

```
42  void CGui::on_pbEnvoyer_clicked()
43  {
44      CModBusRTU _mb;
45      _mb.formerRequete(0x02, 0x03, 0x0000, 0x0001);
46      _mb.getTrame(_trame);
47      // envoi de la trame
48      _com->write( (char *)_trame, 8);
49      if (_com->waitForReadyRead(5000)) {
50          _reponse = _com->readAll();
51          qDebug() << "Recu:" << _reponse;
52          int nbo = _reponse.size();
53          qDebug() << "size réponse : " << nbo;
54          qDebug() << "CRChigh : " << (int)(_reponse.at(nbo-1)) << " / CRClow : " << (int)
55          _crc = ((_reponse.at(nbo-1)<<8) & 0xff00) | (_reponse.at(nbo-2) & 0x00ff);
56          qDebug() << "crc reçu : " << _crc;
57          _mb.setTrame(_reponse);
58
59          if (_mb.comparerCrC(_crc)){
60              ui->statusbar->showMessage("Good CRC");
61              _data = _mb.extraireDatasReponse();
62              uint16_t dirVent;
63              dirVent = (_data.at(0)<<8) | _data.at(1);
64              float reelVent = dirVent/10;
65              ui->leRep->setText(QString::number(reelVent,'g',1));
66              ui->statusbar->showMessage("Tout va bien");
67          } else {
68              qDebug() << "CRC16 mauvais !";
69              ui->statusbar->showMessage("CRC16 mauvais !");
70          } // else
71      } else {
72          qDebug() << "Pas de réponse !";
73          ui->statusbar->showMessage("Pb réception");
74      } // else
75  } // methode
76
```

Ce programme permet d'acquérir les données que renvoie la girouette à chaque fois que l'on appuie sur le bouton de l'interface, et de l'afficher sur une fenêtre. Si le crc est mauvais, un code d'erreur est affiché, et la même chose survient si l'on ne réussit pas à se connecter au port série.

L’anémomètre.

Désormais, nous pouvons poursuivre avec l’anémomètre. Le programme utilisé pour l’acquisition des données qu’il transmet est pratiquement le même que celui utilisé pour la girouette, et cela parce qu’il emploie aussi le protocole modbus.

La classe CAnemometre hérite elle aussi de la classe CModBusRTU.

La différence vient de l’adresse de l’esclave, qui se doit d’être différente.

On a décidé pour le moment de choisir l’adresse 0x03.

Afin de paramétrer l’adresse de l’esclave, il n’y a rien de plus simple. Selon le fournisseur on doit envoyer la trame suivante à l’appareil à l’aide d’un logiciel tel que termite ou commix:

Slave Add	Function Code	Register Start Add	Length of Register	Number of Valid Bytes	Slave Address Written-in	High Bit of Check Code	Low Bit of Check Code
1byt	1byt	2byt	2byt	1byt	2byt	1byt	1byt
0x00	0x10	0x10 0x00	0x00 0x01	0x02	0x00 0x03	FA	00

Contrairement à 8 octets, on a ici une requête de 11 octets afin de modifier l’adresse de l’anémomètre. Le premier octet, qui normalement désigne l’adresse de l’esclave, est ici 0x00. Cette adresse est l’adresse de broadcast en Modbus. Il est intéressant de noter que l’adresse de l’esclave est contenue dans le registre 0x1000.

Les octets qui nous intéressent vraiment sont le 8ème et 9ème, vu que ce sont ces octets qui vont permettre de modifier l’adresse.

Une fois ceci effectuer, pour assurer la réussite de la modification, on reçoit la trame suivante:

00 10 10 00 00 01 04 D8 (sur 7 octets).

Voici donc l’unique changement comparé à Cgirouette.

Lors de l’emploie de la methode formerRequete, on forme donc cette ligne :

formerRequete(0x03,0x03,0x0000,0x0001);

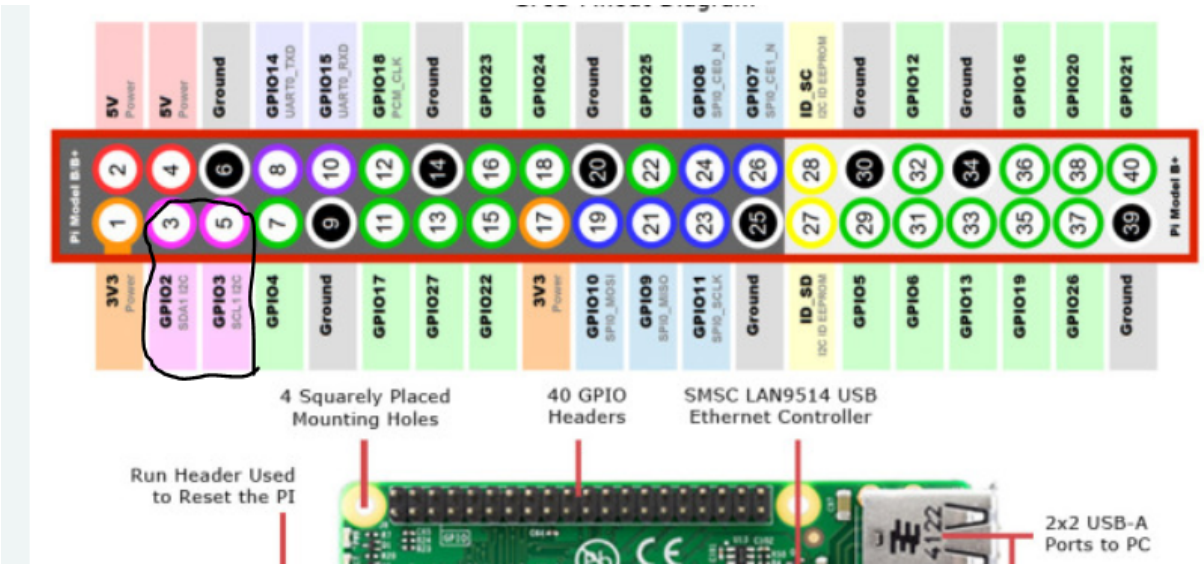
Le capteur angulaire.

La prochaine étape était de créer le début du programme permettant de recevoir les données que renvoie le capteur angulaire. Malgré le manque d'information que j'avais sur celui-ci sur le moment, il nous était informé que le capteur aller employer le protocole de communication I2C.

Mon professeur, Mr Antoine, à créer sa propre librairie I2C, du nom de CI2C, ce qui facilite grandement les choses.

Mais avant de programmer, on configure la carte Raspberry afin qu'elle puisse communiquer en I2C.

Les signaux SDA et SCL se trouvent respectivement sur la broche 3 et la broche 5 de la Raspberry.



Un peu d’explication sur le fonctionnement de l’I2C.

Le SDA et le SCL sont les deux lignes de communication qui sont utilisés pour communiquer entre des composants électroniques dans un système en I2C.

Le SDA (Serial Data) est utilisé pour transmettre les données entre les composants, tandis que le SCL (Serial Clock) est utilisé pour synchroniser la transmission des données entre les composants.

Le SCL est la ligne d'horloge série utilisée pour synchroniser la transmission des données sur la ligne SDA. Le SCL est une ligne unidirectionnelle qui est générée par le maître I2C et utilisée par tous les périphériques pour synchroniser les transferts de données.

On télécharge donc les pacquets suivants :
sudo apt-get install -y python-smbus
sudo apt-get install -y i2c-tools

Et à l’aide de l’outil i2cdetect, on peut identifier les adresse des modules I2C connectés à la carte Raspberry, ce qui va etre essentiel afin de vérifier que le capteur soit bien connecter.

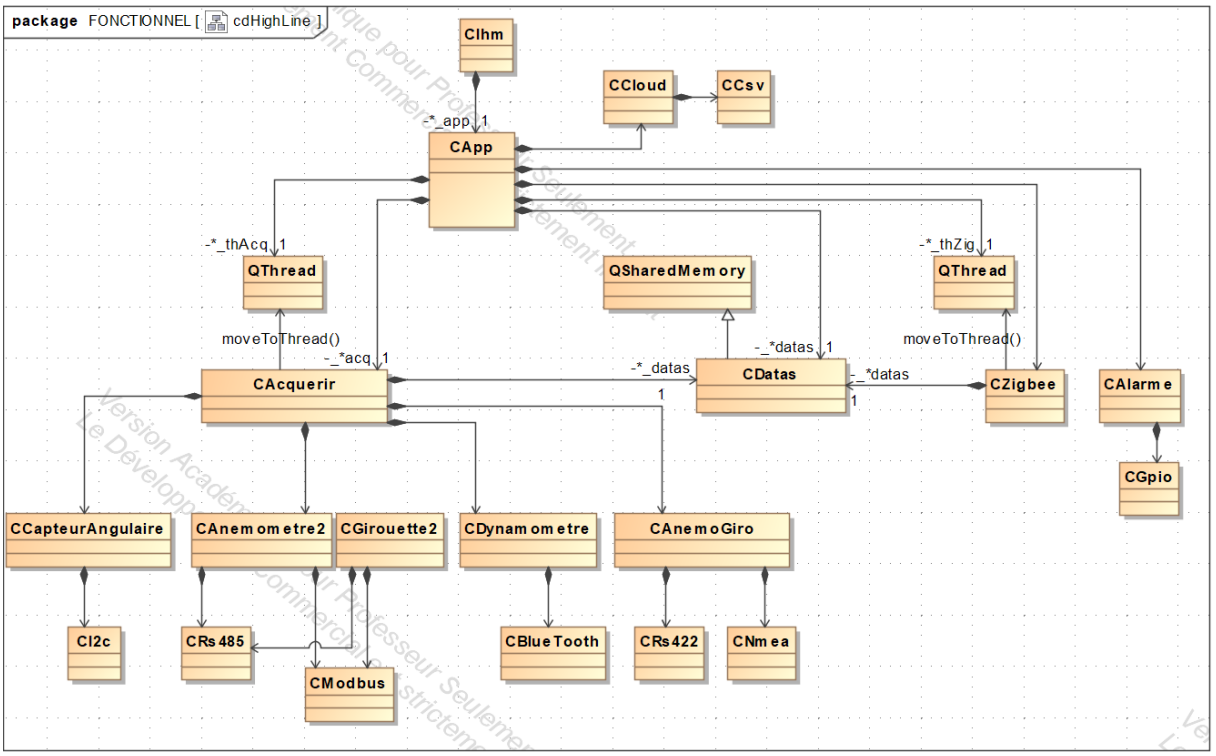
Après des nouvelles directives, on a tous appris que le capteur allait finalement employer le protocole modbus.

Le programme devient alors extrêmement similaire aux autres, si ce n’est pour l’adresse de l’esclave.

Comme pour l’anémomètre, on envoie la trame de modification, cette fois en indiquant les octets suivants pour lui attribuer l’adresse 04 : 0x00 0x04.

La librairie Ci2c est quand même sauvegardé, vu qu’il est largement possible qu’on est à employer de l’I2C dans la suite du projet.

Le partage des données sur le programme.



Comme on le voit sur le diagramme, une classe du nom de CDatas va hériter de la classe QSharedMemory.

L'utilité de la classe QSharedMemory réside dans sa capacité à permettre à plusieurs processus, ou des threads, de partager des données de manière efficace et rapide. Cette classe permettra aux différentes autres classes du programme d'exploiter toutes les données des différents capteurs.

Alors dans un premier temps, dans la classe CDatas, nous allons créer un thread de CAcquerir, à l'aide de la classe QThread afin de directement récupérer toutes les données que renvoient les classes des capteurs.

Qu’est-ce qu’un thread ?

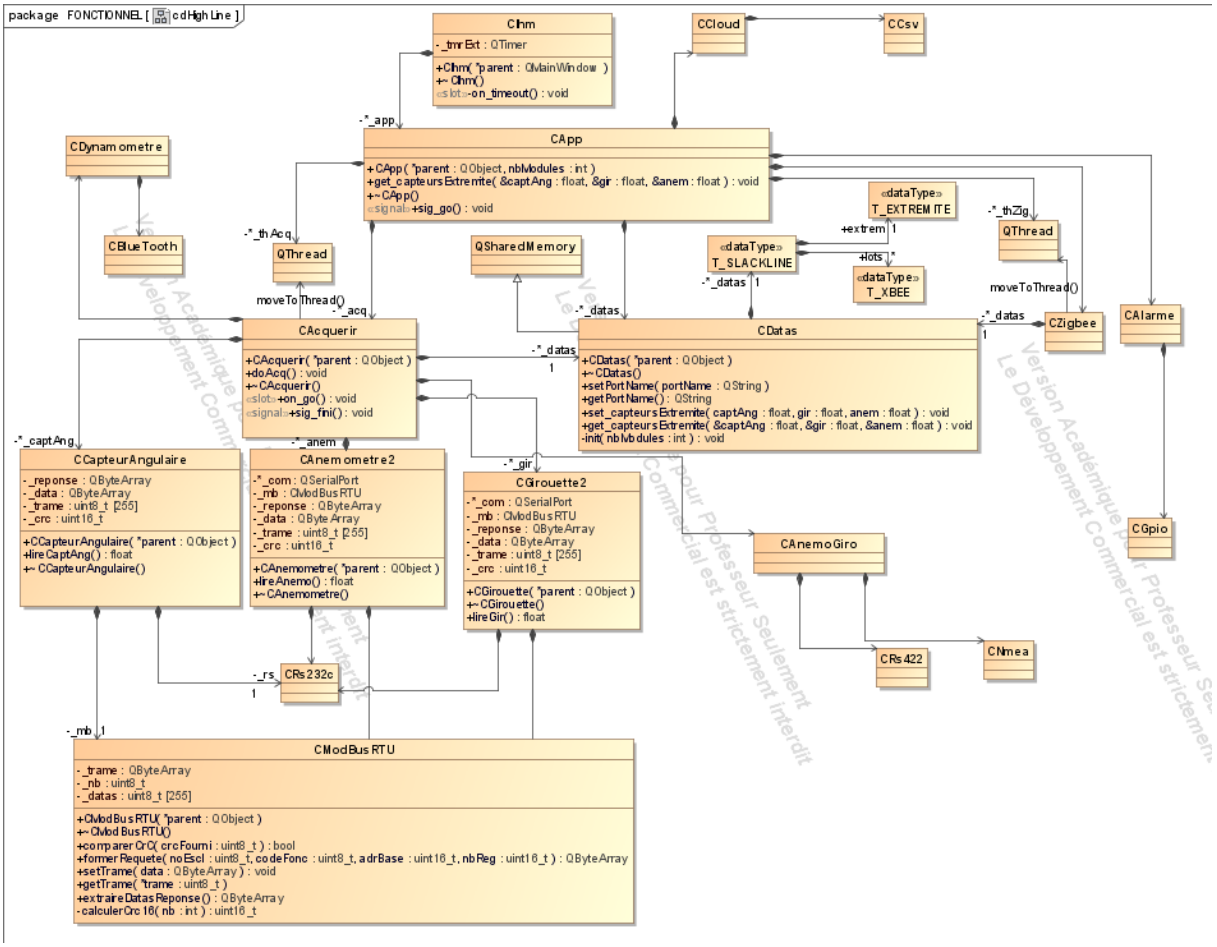
En C++, un thread est un flux d'exécution indépendant qui peut s'exécuter simultanément avec d'autres threads au sein d'un même processus. Les threads sont utiles pour exécuter des tâches en parallèle afin d'augmenter la performance du programme.

Les threads peuvent communiquer et partager des données entre eux, mais cela nécessite une gestion soigneuse pour éviter les problèmes de concurrence et de synchronisation. En C++, les verrous (`mutex`) et les variables conditionnelles (`condition_variable`) sont utilisés pour synchroniser l'accès aux données partagées entre les threads.

En bref, les threads permettent de réaliser plusieurs tâches en même temps sans nécessiter la création de plusieurs processus séparés. Cette approche peut accélérer les temps d'exécution et améliorer les performances globales d'un programme.

L'utilisation de la classe `QThread` facilite la gestion des threads dans les programmes en offrant des méthodes et des signaux pour contrôler et surveiller l'état des threads.

Nous sommes 2 élèves à nous occuper de la zone de mémoire commune, moi et Maxence MERCHAT, qui s’occupe de la partie Zigbee du système.


package FONCTIONNEL [cdHighLine]
Chm
- _tmrExt : QTimer
+Chm(*parent : QMainWindow)
+~Chm()
«slot»-on_timeout() : void
CCloud
CCsv
CApp
+CApp(*parent : QObject, nbIModules : int)
+get_capteursExtremite(&captAng : float, &gir : float, &anem : float) : void
+~CApp()
«signal»+sig_go() : void
CDynamometre
CBlueTooth
QThread
QSharedMemory
«dataType» T_EXTREMITE
«dataType» T_SLACKLINE
«dataType» T_XBEE
CAcquerir
+CAcquerir(*parent : QObject)
+doAcq() : void
+~CAcquerir()
«slot»+on_go() : void
«signal»+sig_fini() : void
CDatas
+CDatas(*parent : QObject)
+~CDatas()
+setPortName(portName : QString)
+getPortName() : QString
+set_capteursExtremite(captAng : float, gir : float, anem : float) : void
+get_capteursExtremite(&captAng : float, &gir : float, &anem : float) : void
-init(nbIModules : int) : void
CZigbee
CAlarme
CGpio
CCapteurAngulaire
- _reponse : QByteArray
- _data : QByteArray
- _trame : uint8_t [255]
- _crc : uint16_t
+CCapteurAngulaire(*parent : QObject)
+lireCaptAng() : float
+~CCapteurAngulaire()
CAnemometre2
-*_com : QSerialPort
- _mb : CModBusRTU
- _reponse : QByteArray
- _data : QByteArray
- _trame : uint8_t [255]
- _crc : uint16_t
+CAnemometre(*parent : QObject)
+lireAnemo() : float
+~CAnemometre()
CGirouette2
-*_com : QSerialPort
- _mb : CModBusRTU
- _reponse : QByteArray
- _data : QByteArray
- _trame : uint8_t [255]
- _crc : uint16_t
+CGirouette(*parent : QObject)
+~CGirouette()
+lireGir() : float
CAnemoGiro
CRs232c
CRs422
CNmea
CModBusRTU
- _trame : QByteArray
- _nb : uint8_t
- _datas : uint8_t [255]
+CModBusRTU(*parent : QObject)
+~CModBusRTU()
+comparerCrC(crcFourni : uint8_t) : bool
+formerRequete(noEscl : uint8_t, codeFonc : uint8_t, adrBase : uint16_t, nbReg : uint16_t) : QByteArray
+setTrame(data : QByteArray) : void
+getTrame(*trame : uint8_t)
+extraireDatasReponse() : QByteArray
-calculerCrc16(nb : int) : uint16_t
-*_app
-*_thAcq
-*_thZig
-*_acq
-*_datas
-*_anem
-*_gir
-*_captAng
-*_rs
-_mb
moveToThread()
+extrem
+lots
Version Académique pour Professeur Seulement
Le Développement Commercial est strictement interdit

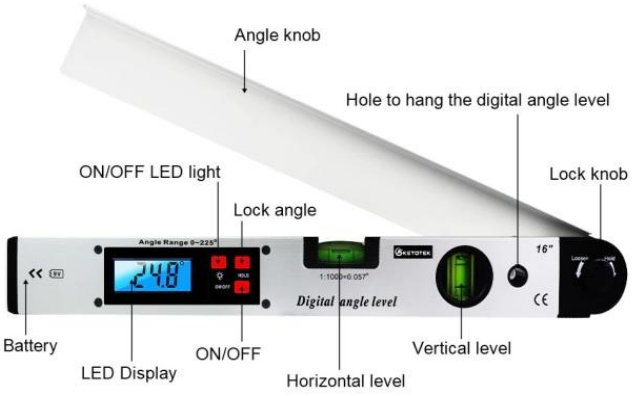
Partie individuelle
Fleury Benjamin

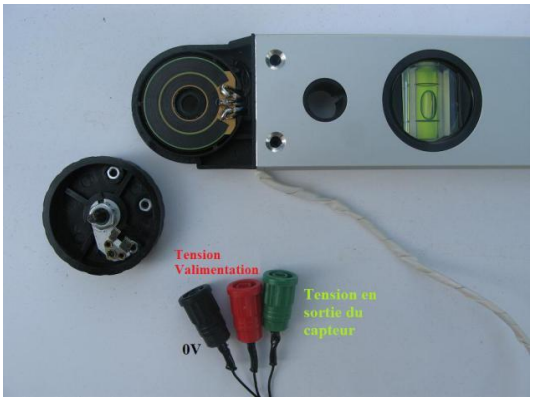
Pour ma partie du projet je dois mettre en œuvre, grâce à des codes sur arduino, un capteur angulaire allant de 0° à 300°, ainsi qu'un avertisseur sonore qui s'activera à un angle défini, afin de mesurer l'angle des cordes pour que les tensions sur celles-ci soient réparties uniformément pour une meilleur stabilité et une meilleur sécurité. Je devrai ensuite envoyer les mesures obtenues sur une raspberry pi par liaison MODBUS.
Pour le réaliser j'ai à ma disposition:
-Un **ESP32-S3 T-Display LILYGO** qui est un écran programmable qui me servira à afficher l'angle en temps réel.
-Un rapporteur numérique que j'utilise afin de comprendre comment ça fonctionne et pour m'en servir de modèle.
-Un potentiomètre linéaire pour les tests.
-Une carte arduino uno.
-Le logiciel arduino.

DIAGRAMME DE GANTT

		Mod Tâch	Nom de la tâche	Début	Fin	Durée	Prédécesseurs
10			**Projet**	**Mar 03/01/23**	**Ven 16/06/23**	**257 h?**	
11			Specifications générales et lecture du dossier	Mar 03/01/23	Mer 04/01/23	12 h	
12			Découverte et apprentissage du matériel à utiliser	Lun 09/01/23	Mar 10/01/23	4 h	11
13			Test du rapporteur numérique sur une arduino uno	Mar 10/01/23	Mar 10/01/23	2 h	12
14			Prise en main du LILYGO ESP32 et programmation pour l'affichage sur celui-ci	Mar 10/01/23	Mer 18/01/23	12 h	13
15			Programmation du LILYGO ESP32 pour un potentiomètre et affiché l'angle du potentiomètre sur le moniteur série du logiciel arduino	Mer 18/01/23	Lun 30/01/23	14 h	14
16			Changement de potentiomètre et ajout d'une référence de tension pour que les résultats soient plus précis	Lun 27/02/23	Mar 28/02/23	7 h	15
17			Modification du programme pour affiché la tension et l'angle sur l'écran du LILYGO ESP32 et ajout d'un CAN pour y parvenir	Lun 06/03/23	Mer 15/03/23	20 h	16
18			Divers essais avec des multimètres et d'autres potentiomètres pour plus de précision	Lun 20/03/23	Mar 04/04/23	24 h	17
19			Routage de la carte	Mar 21/03/23	Mer 12/04/23	41 h	
20			Fabrication et essais	Mar 02/05/23	Mer 07/06/23	57 h	19

Après avoir pris connaissance du dossier, j'ai commencé à regarder les spécifications du rapporteur numérique qui fonctionne comme un potentiomètre j'ai ensuite fait un programme sur une arduino uno permettant de vérifier l'angle du rapporteur numérique ainsi que la tension, par exemple lorsque le rapporteur numérique est à un angle de 115° il y'a une tension de 2.5V, qui a été affichée sur le moniteur série.





Capture d'écran du code arduino permettant l'affichage de l'angle et le tension sur le moniteur série:

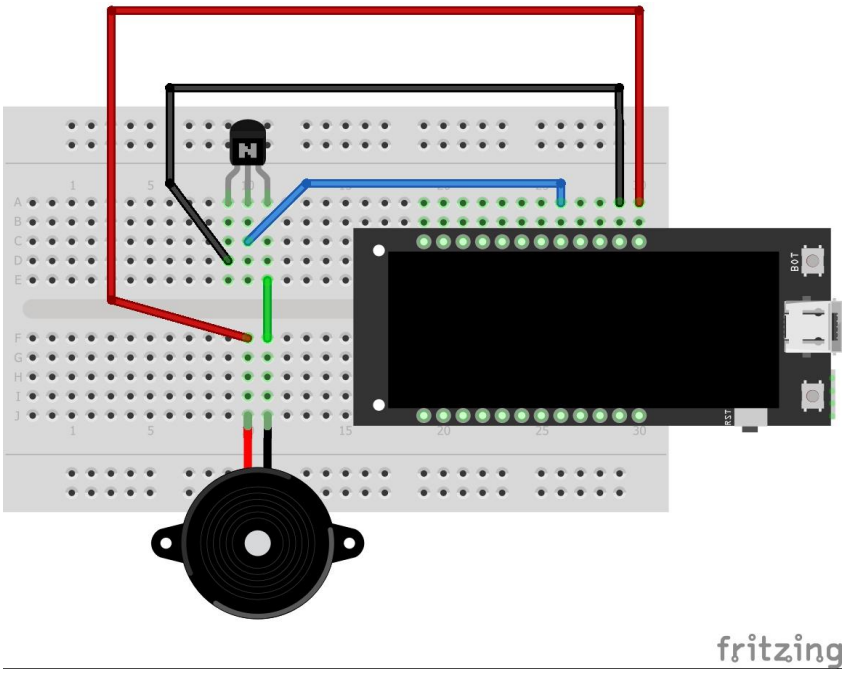
```
int sensorPin = A0;    // select the input pin for the potentiometer
int ledPin = 13;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor
float tensionImageAngle;
float angleMaximal = 291.0;
float angleMesure;
int angle;

void setup() {
  Serial.begin(9600);
  Serial.println("Mesure angle rapporteur numérique");
  Serial.println("");
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
//  tensionImageAngle = ((sensorValue-15) * 5.1)/1023.0;
  tensionImageAngle = ((sensorValue-15) * 5.23)/1023.0;
//  angleMesure = ((sensorValue-15) * (angleMaximal*1.02))/1023.0;
  angleMesure = ((sensorValue-15) * (angleMaximal*1.046))/1023.0;
  angle = round(angleMesure);
  Serial.print("tension angle = ");
  Serial.print(tensionImageAngle);
  Serial.print("V / Angle mesuré = ");
  Serial.print(angleMesure);
  Serial.print(" degrés / Angle arrondi = ");
  Serial.print(angle);
  Serial.println(" degrés");
  delay(500);
}
```

J’ai ensuite pris connaissance des vidéos et diverses documentations concernant l’ **ESP32-S3 T-Display LILYGO** et fait un programme pour mettre une image sur son écran. Capture d'écran du code arduino permettant l’affichage sur l’ **ESP32-S3 T-Display LILYGO**:

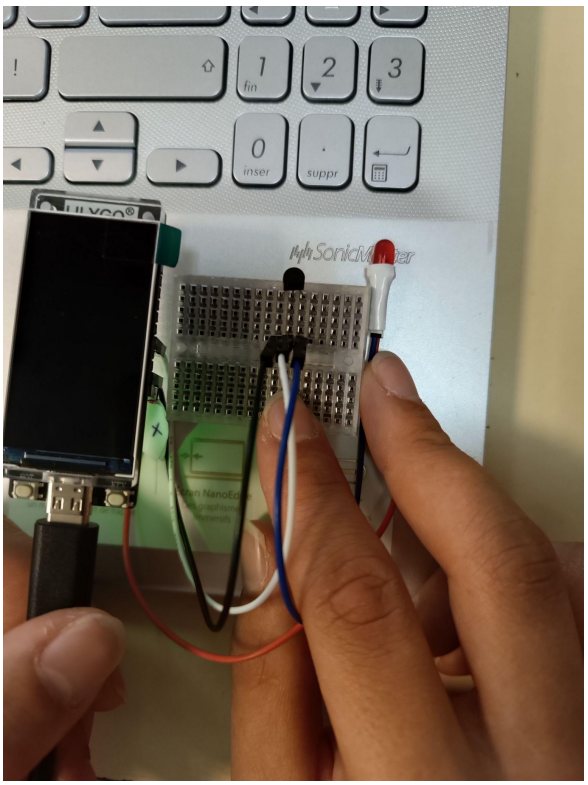
```
#include "TFT_eSPI.h"
#include "hello_world.h"

TFT_eSPI tft= TFT_eSPI();

void setup() {
  tft.init();
  tft.setRotation(3);
  tft.setSwapBytes(true);
  tft.fillScreen(TFT_BLACK);
  tft.pushImage(0,80,320,88,hello_world_320x88); // Rappel : Taille Ecran 320x170
}


void loop() {
  tft.setTextDatum(0);
  tft.setTextColor(TFT_NAVY);
  tft.drawString("Test Lilygo", 0, 0, 2);
  tft.setTextColor(TFT_VIOLET);
  tft.drawString("Test Lilygo", 160, 40, 4);
  delay(1000);
}
```

En modifiant les lignes setTextColor et drawString nous pouvons écrire ce que l’on souhaite et de la couleur que l’on souhaite et avec la ligne pushImage nous pouvons mettre une image en arrière plan, il faut que l’image soit à la taille de l'écran sinon ça ne fonctionnera pas. Pour cela il faut aller sur un site de convertisseur d’image pour la mettre au format de l’afficheur.

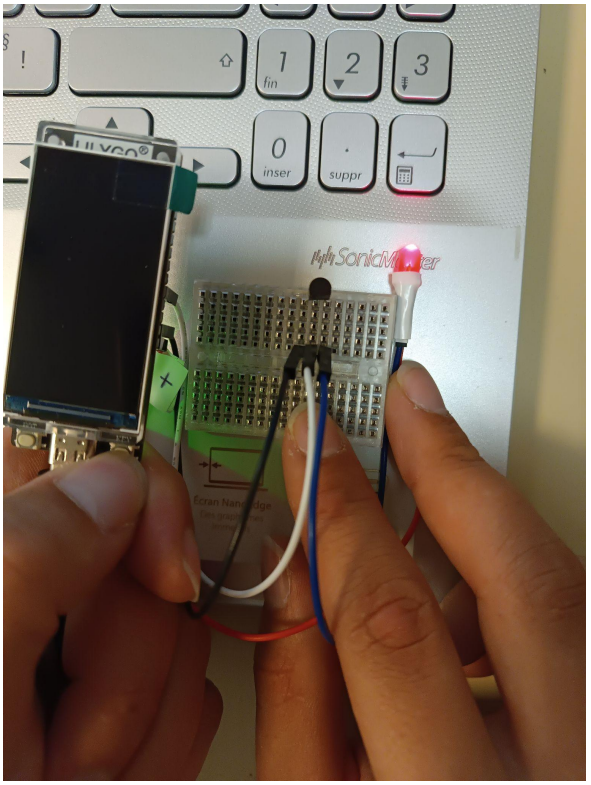
L' **ESP32-S3 T-Display LILYGO** possède également un bouton poussoir programmable, que j'ai programmer pour que lorsque que l'on appui dessus un buzzer se mette à sonner et s'arrête quand on le relâche, voici le schéma fritzing du câblage sur lequel se trouve le buzzer,l' **ESP32-S3 T-Display LILYGO**, un transistor mosfet VN2222LL:



Voici des photos du montage non pas avec un buzzer mais une LED, le montage et le programme sont les même:



Bouton relâché



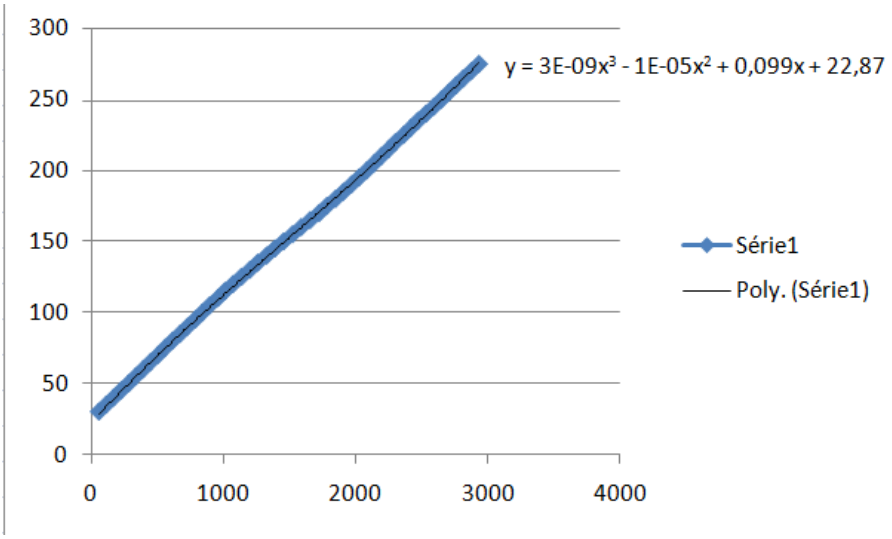
Bouton appuyé

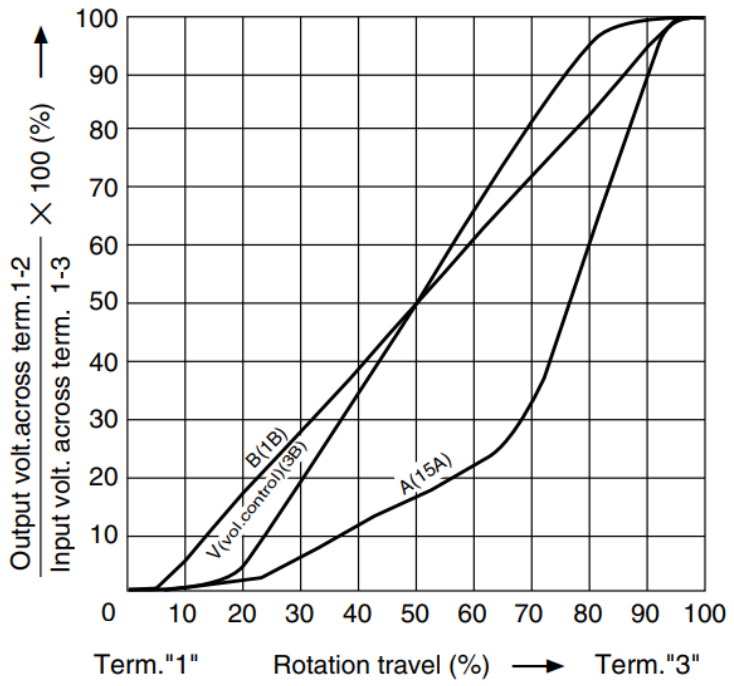
Voici le programme permettant de programmer le bouton poussoir sur le logiciel arduino:

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 14;     // the number of the pushbutton pin
const int BuzzPin =  13;      // the number of the BUZZ pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the BUZZ pin as an output:
  pinMode(BuzzPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == LOW) {
    // turn BUZZ off:
    digitalWrite(BuzzPin, HIGH);
    Serial.println("BP Key appuyé");
  } else {
    // turn BUZZ on:
    digitalWrite(BuzzPin, LOW);
    Serial.println(" BP Key relâché");
  delay(100);
  }
}
```

En modifiant le programme arduino du rapporteur numérique j'ai pu programmer l' **ESP32-S3 T-Display LILYGO** pour qu'il puisse mesurer l'angle et la tension d'un potentiomètre mais lors des premiers essais les résultats n'étaient pas concluant, il y avait une plage au début du potentiomètre ou lorsque l'on tournait il n'y avait aucun angle et aucune tension et à la fin du potentiomètre sur 1/8 de la rotation l'angle et la tension ne changeait pas.

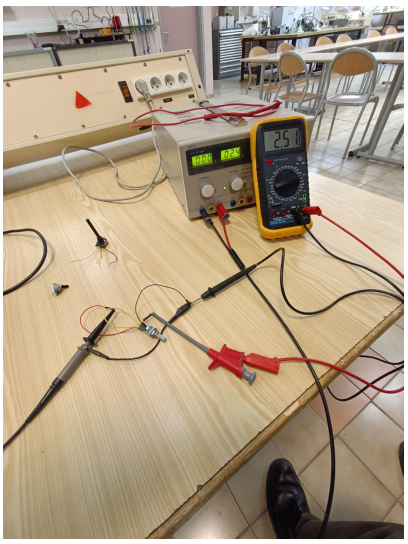
TENSION	ANGLE
62	30
118	35
174	40
230	45
288	50
342	55
398	60
452	65
510	70
562	75
619	80
677	85
733	90
792	95
846	100
901	105
962	110
1016	115
1077	120
1140	125
1204	130
1264	135
1331	140
1393	145
1460	150
1527	155
1592	160
1660	165
1728	170
1787	175
1852	180
1912	185
1971	190
2029	195
2088	200
2144	205
2198	210
2254	215
2306	220
2367	225
2422	230
2476	235
2537	240
2597	245
2654	250
2708	255
2764	260
2820	265
2879	270
2937	275

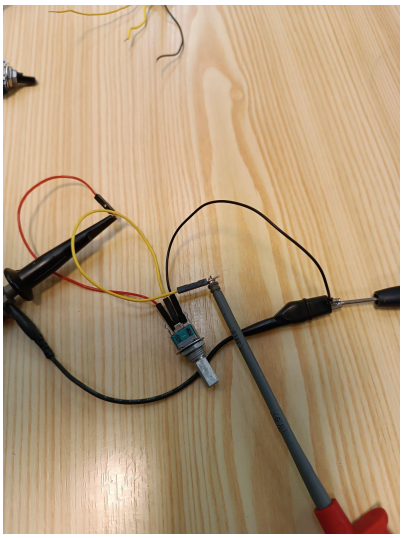


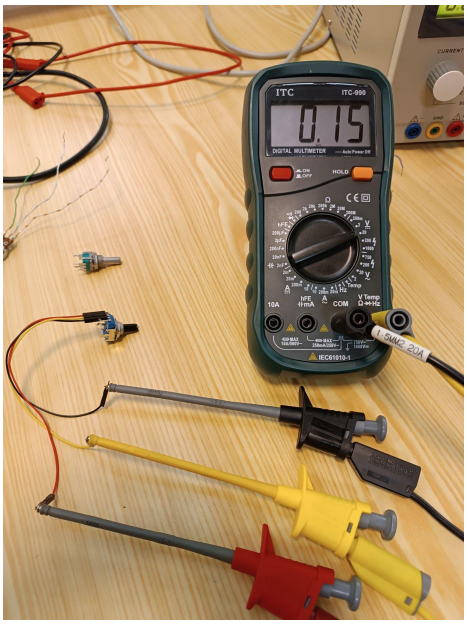


Ci-dessus se trouve quatres images, les trois premières représentent une courbe calculée pour le potentiomètre et sur la dernière se trouve un extrait de la documentation technique du potentiomètre.

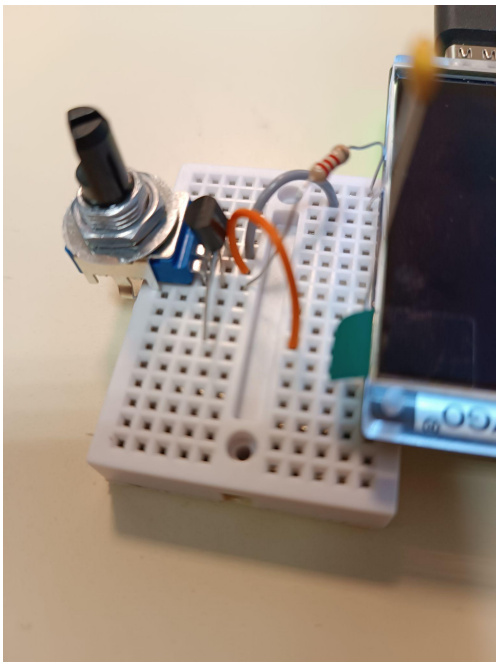
Suite à ces essais j'ai changer de potentiomètre car celui que j'avais était équipé de ce qu'on appelle "center detent" ce qui signifie que lorsque l'on tourne le potentiomètre il y'a un cran d'arrêt au milieu ce qui empêche la bonne prise en compte des mesures mais il y'a eu des problèmes avec le nouveau potentiomètre car de 0° à 25° la tension et l'angle sur l'afficheur était de 0° la tension sur le multimètre (branché pour les vérifications) afficher 0V. Les tests ont été réalisés avec d'autres potentiomètre et avec une alimentation extérieure, le résultat était le même pour 2 d'entre eux 2 autres ne fonctionnaient tout simplement pas.



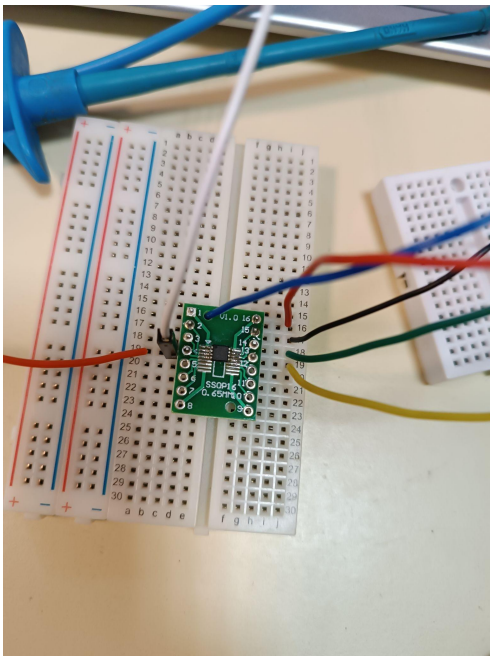




Etant donné que le problème était toujours présent, monsieur Hortolland m'a fait faire un montage différent avec une référence de tension de 2,5V, un LM4040, et une résistance. Le résultat était mieux mais toujours pas assez précis.

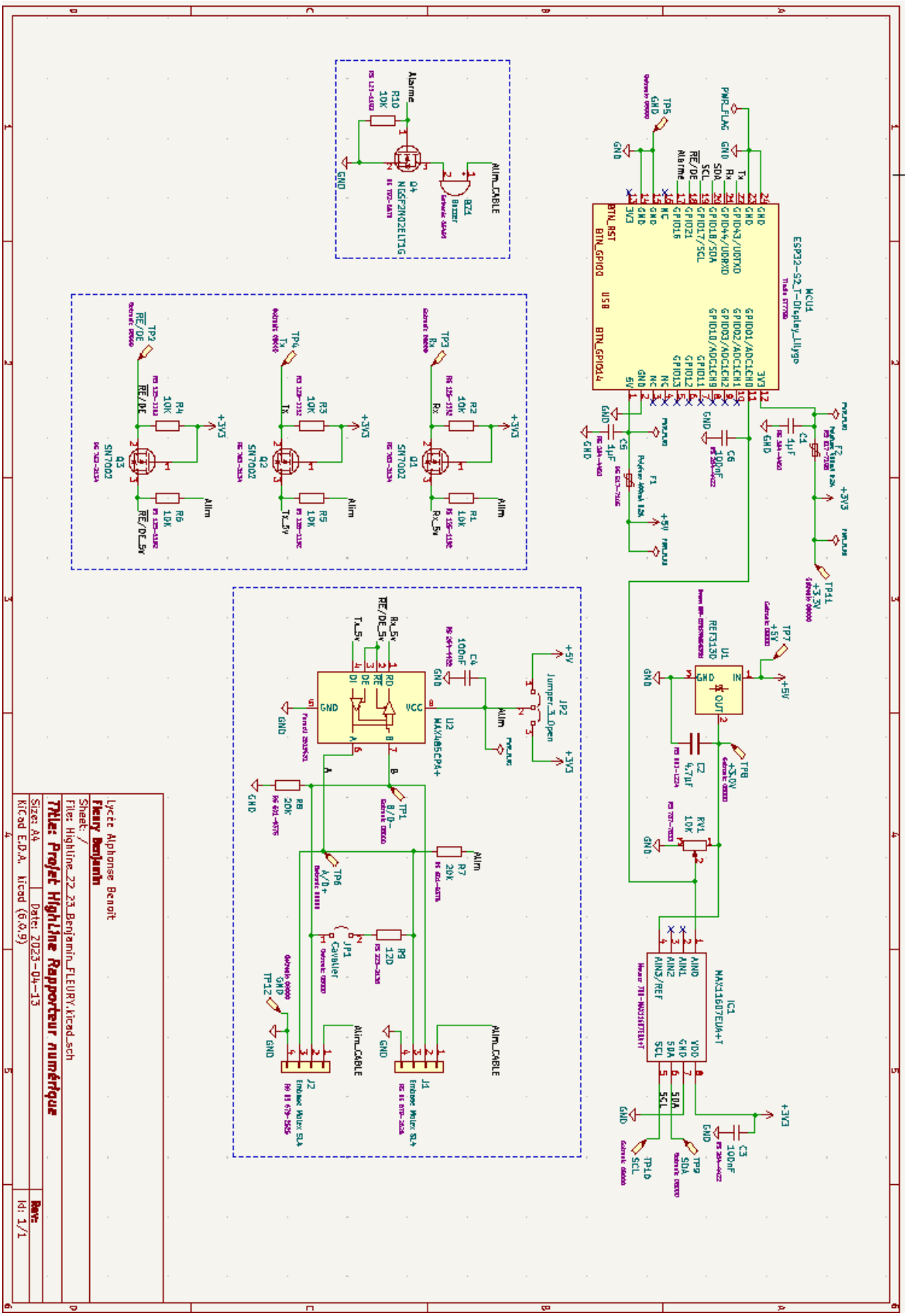


Un autre montage à été réalisé cette fois sans la résistance et avec un condensateur entre la masse et l'entrée du GPIO et à la place du LM4040 un MAX11607 a été mis et le résultat est beaucoup plus précis.

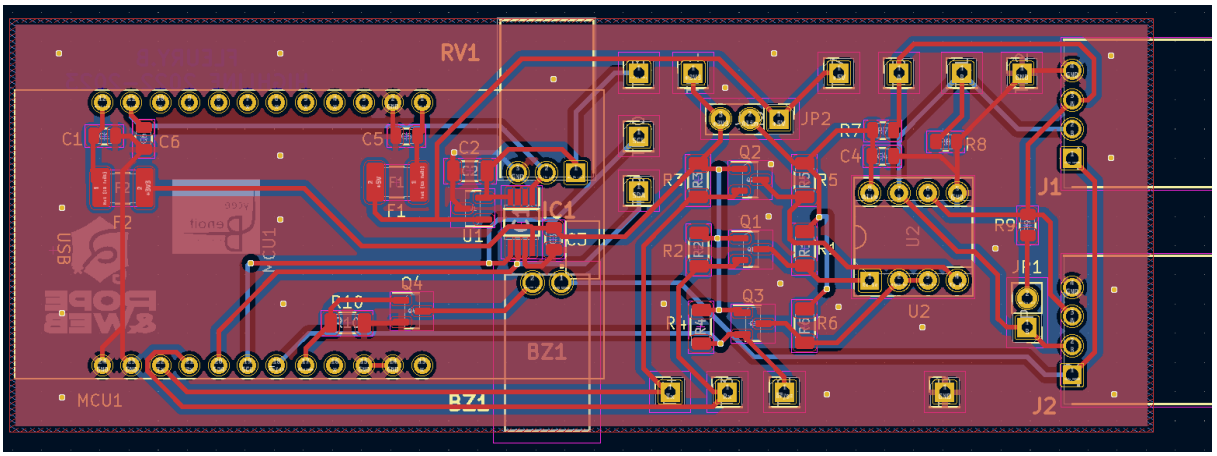


Une carte à été routée avec tous les composants et des emplacements pour le buzzer et le potentiomètre.

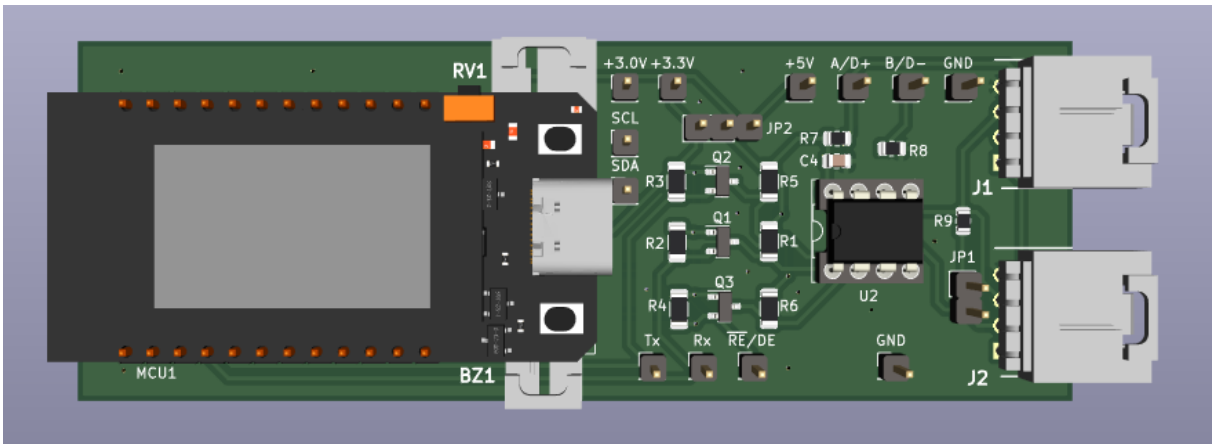
Schéma complet de la carte:


Lycée Alphonse Benoit
Fleury Benjamin
Sheet: /
File: Highline_22_23_Benjamin_FLEURY.kicad_sch
Title: Projet Highline Rapporteur numérique
Size: A4
Date: 2023-04-13
KiCad E.D.A. kicad (6.0.9)
Rev:
Id: 1/1
MCU1
ESP32-S2_T-Display_Liligo
IC1
MAX11607EUA+T
U1
REF3130
U2
MAX14855CPA+
Alarme
Alim_CABLE
RE/DE_5v
Rx_5v
Tx_5v
Alim
+3V3
+5V
GND
SDA
SCL

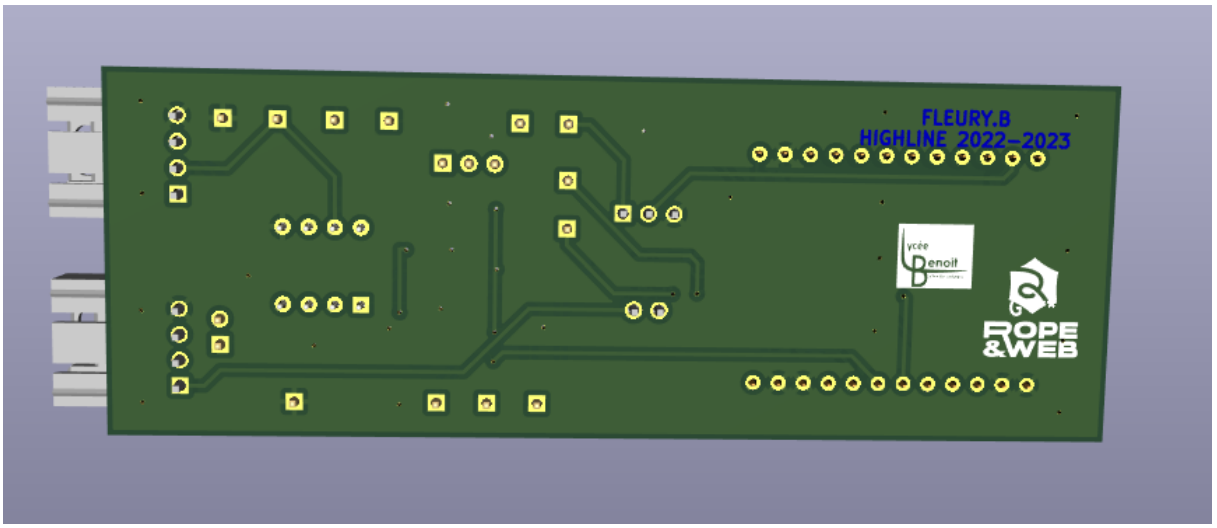
Vue top de la carte:



Vue en 3D top de la carte:



Vue en 3D bottom de la carte:



Liste des composants à souder sur la carte:

	Valeur	Code Commande	Quantités
BZ1	Buzzer	Gotronic 05460	1
C2	4,7µF	RS 691-1224	1
> C1, C5	1µF	RS 264-4450	2
> C3, C4, C6	100nF	RS 264-4422	3
> F1, F2	Polyfuse 400mA 0,2A	RS 517-7105	2
IC1	MAX11607EUA+T	Mouser 700-MAX11607EUA+T	1
> J1, J2	Embase Molex SL4	RS RS 679-2525	2
JP1	Cavalier	Gotronic 08000	1
JP2	Jumper_3_Open		1
MCU1	ESP32-S2_T-Display_Lilygo	Tindie ST7789	1
> Q1-Q3	SN7002	RS 753-3134	3
Q4	MGSF2N02ELT1G	RS 792-5675	1
> R1-R6, R10	10K	RS 125-1192	7
> R7, R8	20K	RS 601-9375	2
R9	120	RS 223-2136	1
RV1	10K	RS 737-7833	1
TP1	B/D-	Gotronic 08000	1
TP2	~{RE}/DE	Gotronic 08000	1
TP3	Rx	Gotronic 08000	1
TP4	Tx	Gotronic 08000	1
> TP5, TP12	GND	Gotronic 08000	2
TP6	A/D+	Gotronic 08000	1
TP7	+5V	Gotronic 08000	1
TP8	+3.0V	Gotronic 08000	1
TP9	SDA	Gotronic 08000	1
TP10	SCL	Gotronic 08000	1
TP11	+3.3V	Gotronic 08000	1
U1	REF3130	Mouser 595-REF3130AQDBZRQ1	1
U2	MAX485CPA+	Farnell 2519431	1

Etant donné que les composants sont tous du côté top de la carte je vais commencer par souder les résistances et les condensateurs à l'aide du four puis l'embase pour le MAX11607EUA+T après ça je vais souder les quatre connecteurs et ensuite tous les points de test ainsi que les embases pour l' **ESP32-S3 T-Display LILYGO**.

Je dois communiquer les mesures d'angles à la Raspberry Pi de RICARD Loic.

Partie individuelle
Spina Rémi
Mesure de force (poids) sur mâts de déport

Pour assurer la sécurité du Highliner et plus largement pour l'installation en sécurité de matériel de secours, je suis en charge de mesurer et de communiquer sous forme visuelle et sous forme de données numériques, la valeur des efforts qui vont s'exercer sur les mâts de déport (jusqu'à 300kg), ainsi que déclencher une alarme en cas de dépassement d'un seuil.
Pour réaliser cette tâche, je dispose :
- d'une carte Arduino UNO,
- d'une carte Raspberry,
- d'un capteur de force 50kg SFE avec sa carte d'amplification à base de HX711.

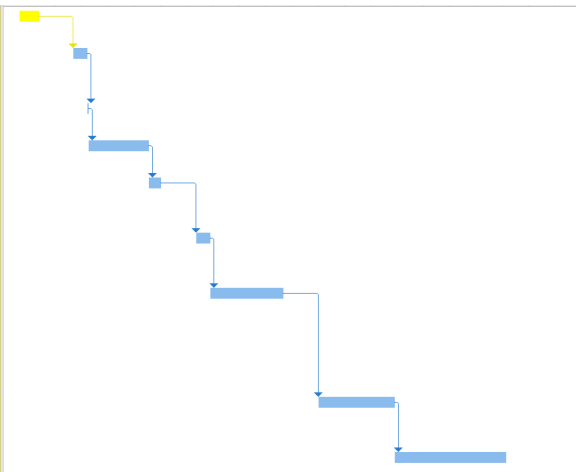
Je vais également m'intéresser à la carte ESP32-S3 T-Display LILYGO sur laquelle le développement final est prévu, elle servira de support visuel, puis assurera en temps qu'esclave la communication d'une liaison MODBUS RTU RS485.
Je concevrai ensuite une carte avec tous ces éléments pour répondre à la problématique.

Diagramme de Gantt :

Ce j'avais prévu :

DIAGRAMME DE GANTT

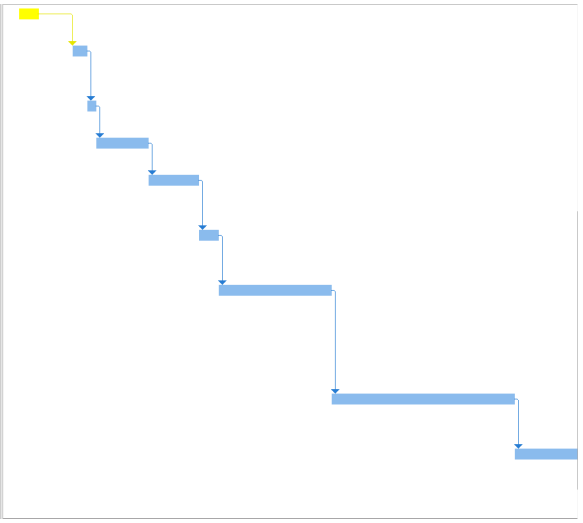
10		Specifications générales	Mar 03/01/23	Mer 04/01/23	12 hr	
11		Documentation sur la transmission MODBUS RTU	Lun 09/01/23	Mar 10/01/23	4 hr	10
12		Apretissage sur les jauges de contraintes	Mar 10/01/23	Mar 10/01/23	2 hr	11
13		Mise en œuvre des capteurs proposés	Mar 10/01/23	Mar 17/01/23	8 hr	12
14		Prendre en main l'outil de développement ESP32S3	Mar 17/01/23	Mer 18/01/23	6 hr	13
15		Transposer le travail effectué sur Arduino Uno sur la carte ESP32	Lun 23/01/23	Mar 24/01/23	4 hr	14
16		Mettre en œuvre l'une des liaisons sans fil de la carte afin de communiquer la mesure vers un Rpi et/ou un PC.	Mar 24/01/23	Mer 01/02/23	16 hr	15
17		Proposer le schéma structurel d'une carte fille.	Lun 06/02/23	Mar 14/02/23	16 hr	16
18		Router un circuit imprimé.	Mar 14/02/23	Lun 27/02/23	16 hr	17
19		Après réception de la carte, la câbler	Lun 01/05/23	Lun 08/05/23	12 hr	



Ce qu'il s'est vraiment passé :

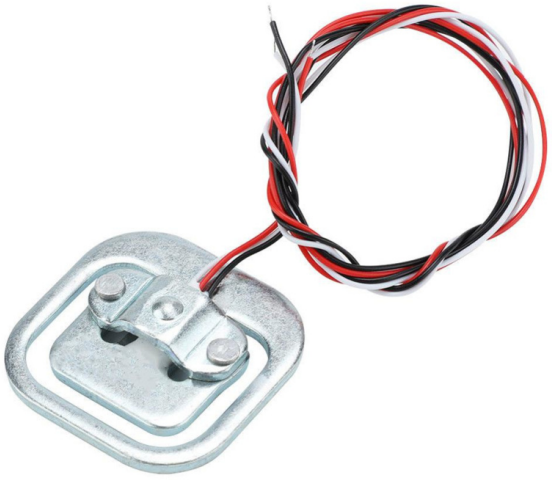
DIAGRAMME DE GANTT

10	Specifications générales	Mar 03/01/23	Mer 04/01/23	12 hr	
11	Documentation sur la transmission MODBUS RTU	Lun 09/01/23	Mar 10/01/23	5 hr	10
12	Apretissage sur les jauges de contraintes	Mar 10/01/23	Mer 11/01/23	3 hr	11
13	Mise en œuvre des capteurs proposés	Mer 11/01/23	Mar 17/01/23	8 hr	12
14	Prendre en main l'outil de développement ESP32S3	Mar 17/01/23	Lun 23/01/23	6 hr	13
15	Transposer le travail effectué sur Arduino Uno sur la carte ESP32	Lun 23/01/23	Mer 25/01/23	7 hr	14
16	Mettre en œuvre l'une des liaisons sans fil de la carte afin de communiquer la mesure vers un Rpi et/ou un PC.	Mer 25/01/23	Mar 07/02/23	16 hr	15
17	Proposer le schéma structurel d'une carte fille.	Mar 07/02/23	Mar 28/02/23	30 hr	16
18	Router un circuit imprimé.	Mar 28/02/23	Lun 13/03/23	16 hr	17
19	Après réception de la carte, la câbler	Lun 01/05/23	Lun 08/05/23	12 hr	



Mise en œuvre des capteurs:

Présentation des capteurs :



Capteur de force 50kg sfe face (otronic.nl)

Capteur de force 50kg sfe profil arrière (otronic.nl)

Le capteur 50kg SFE est un capteur passif qui mesure une force, il est constitué de manière à simuler deux résistance de 1kΩ entre les deux câbles aux extrémités et celui du centre.

Pourquoi ce capteur ?

Pour l'événement, il y aura deux mesures de traction prévue par l'entreprise : Avec un dynamomètre (que je n'aurais pas à effectuer) et avec des loads cells, j'ai été informé, que le

client avait un peu cherché un capteur, et qu'il avait trouvé ces loads cells, car il lui fallait des capteurs pouvant mesurer jusqu'à 300kg (divisé en 3 pieds) qui n'avait pas besoin d'être très precis (2-3kg) pour un prix peu élevé et une bonne practicité. A noter que pour les secours, pour les mâts de déports, nous utiliserons une formation de trois capteurs en triangle, supportant donc une installation tripodes et bipodes mais n'étant pas suffisant pour une formation monopodes, pour résoudre le problème, il suffirait d'une nouvelle formation ou de doubler le nombre de capteurs (par un triangle inversé au premier, par exemple).

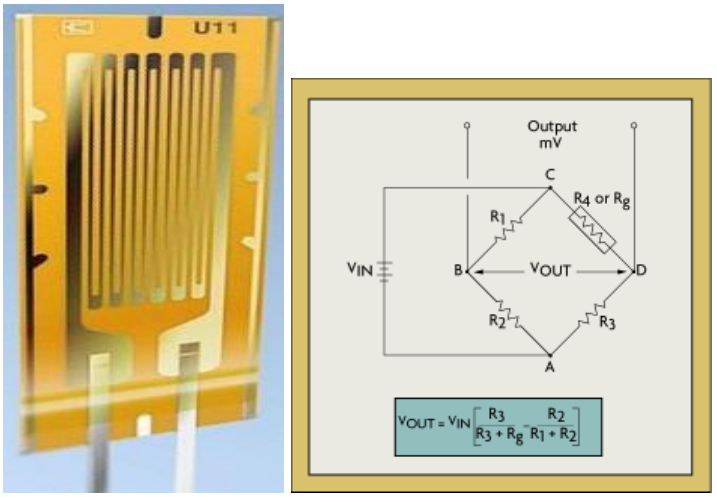
Fiche technique du capteur :

Capacity	kg	40-50
Comprehensive Error	mv/v	0.05
Output Sensitivity	mv/v	1.0±0.1
Nonlinearity	%FS	0.03
Repeatability	%FS	0.03
Hysteresis	%FS	0.03
Creep	(3min)%FS	0.03
Zero Drift	(1min)%FS	0.03
Temp. Effect on Zero	%FS/10℃	1
Temp. Effect on Output	%FS/10℃	0.05
Zero Output	mV/V	±0.1
Input Resistance	Ω	1000±20
Output Resistance	Ω	1000±20
Insulation Resistance	MΩ	≥5000
Excitation Voltage	V	≤10
Operation Temp. Range	℃	0--+50
Overload Capacity	%FS	150

Fonctionnement du capteur :

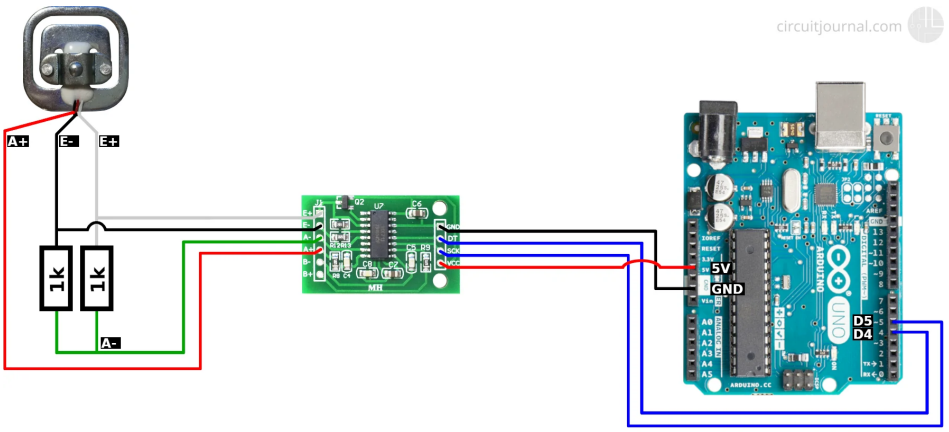
Cette jauge de contrainte fonctionne grâce au principe physique du pont de Wheatstone. Ayant pour but de nous donner la masse et/ou le poids d'un corps physique. Un pont de Wheatstone est un montage en pont de 4 résistances à valeur égales, où lorsqu'on envoie une tension à deux extrémités opposées (A et C par exemple) alors les deux autres extrémités opposées (B et D) indiquent le même potentiel. Le but va être de déséquilibrer le pont via dans notre cas un poids de manière a par exemple compresser R1 et R3 et étendre R2 et R4, ce faisant, la valeur de deux résistances opposés va baisser et celle des deux autres va augmenter, la différence de potentiel va créer une tension proportionnelle. Il ne suffit plus que d'une tare pour donner un sens à cette tension.

Mise en oeuvre des jauges de contraintes

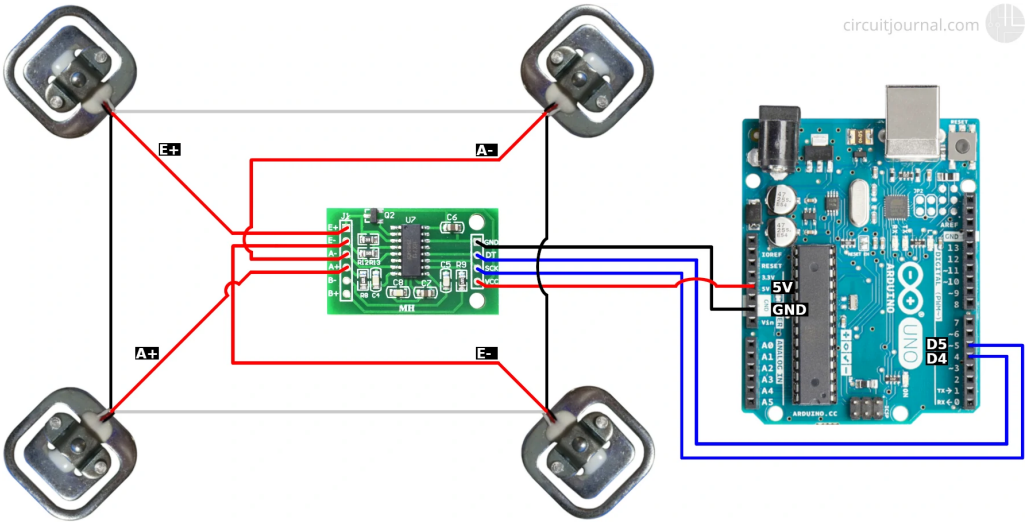


Jauges de contraintes par pei-france.com et pont de Wheatstone par omega.fr > Circuits de mesure

Il y a plusieurs manières de câbler ce capteur, je vais en montrer deux : En premier, je vais parler du branchement d'un seul capteur puis de la combinaison de 4. Ce capteur seul ne peut pas fonctionner, n'ayant que deux résistances, les conditions pour le pont de Wheatstone ne sont pas réunies. Il faut donc brancher deux résistances sur les câbles noirs et blanc qui entourent les résistances et brancher ces résistances à la carte d'amplification :



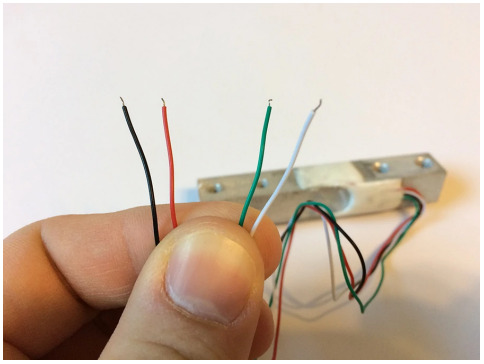
Pour ce qui est des 4 capteurs, il faut brancher les câbles de même couleur entre eux ainsi que de brancher deux câbles rouges opposés, l'un va servir à amener la tension dans le circuit, l'autre va servir à sortir les donnés dû à la modifications de la tension :

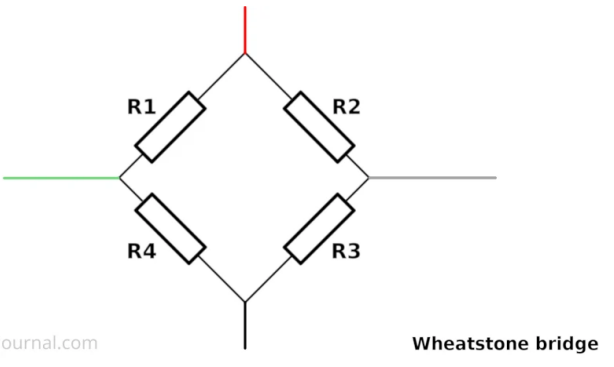


Dans cette figure sachant qu'il y 4 capteur contenant 2 résistances, on pourrait penser à une adaptation nécessaire, il suffit d'interpréter l'addition de la valeur des résistances au bornes du fil d'une même couleur pour se rendre compte qu'on retrouve le schéma à 4 résistances.

Balance de cuisine électronique HX711 5kg :

Ce capteur de force a le même fonctionnement que le précédent. Il est constitué d'une barre métallique avec 4 câbles

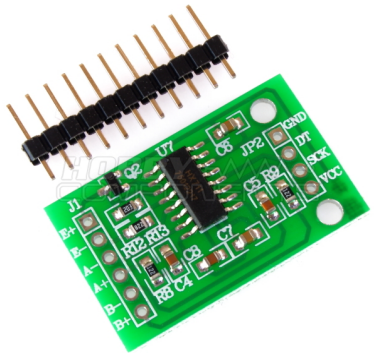


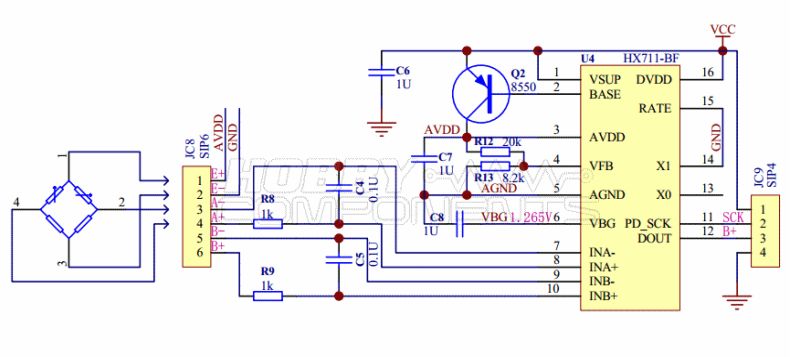


Pour agir sur ce circuit, il suffit d'y apposer un poids sur le cercle en haut, ce qui va déformer la barre.

Carte d'amplification :

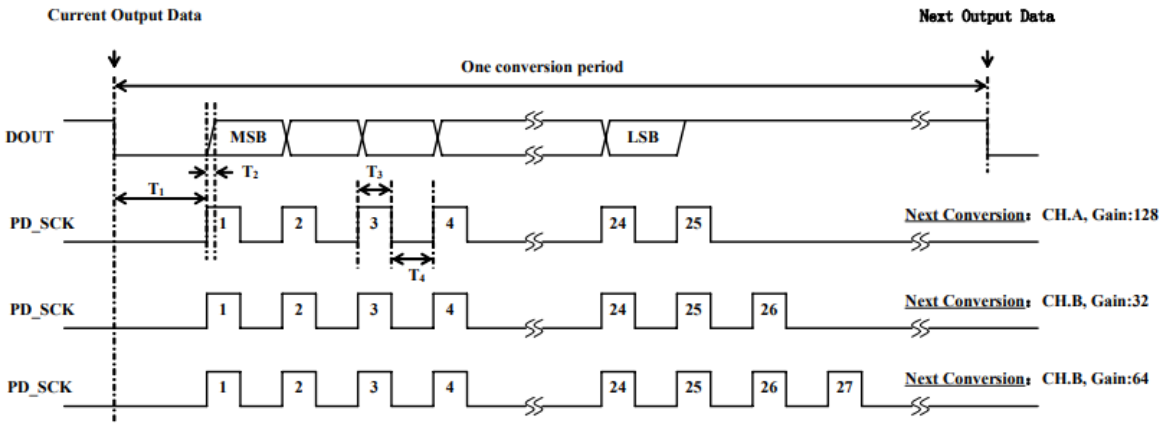
Les deux capteurs présentés ci-dessus ont un problème commun, qui est dû au pont de Wheatstone, c'est simplement que la tension délivrée est trop faible pour être utilisée directement avec un Arduino qui est l'outil que j'utilise pour interpréter les valeurs de ces capteurs. Je vais donc utiliser la carte amplificatrice pour rehausser la tension de manière à ce qu'elle soit compréhensible par l'Arduino. A noter que cette carte est fournie avec les capteurs.

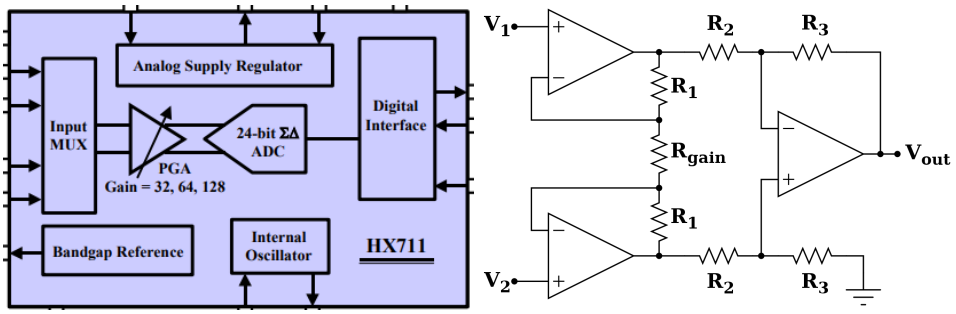




Vis à vis du schéma structurel du breakout ci-dessus, je vais devoir envoyer la tension d'alimentation sur les points E- et E+ et la variation de tension se fera au niveau des points A- et A+. La carte sera alimentée en VCC à +5V (et sous +3.3V dans l'application finale). L'échange avec le microcontrôleur se fera grâce à une liaison série ne fait pas partie des liaisons standard (I2C, SPI, UART) constituée d'un signal d'horloge SCK et d'une donnée DT (Dout).

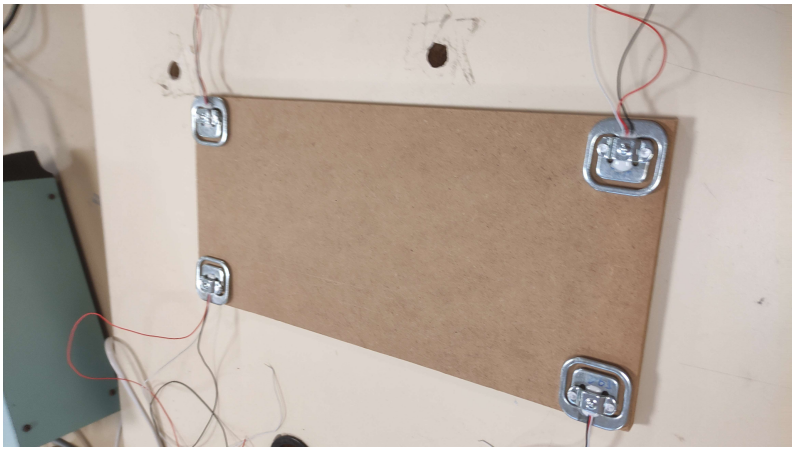
Nous voyons ici un exemple d'une trame de ce protocole :



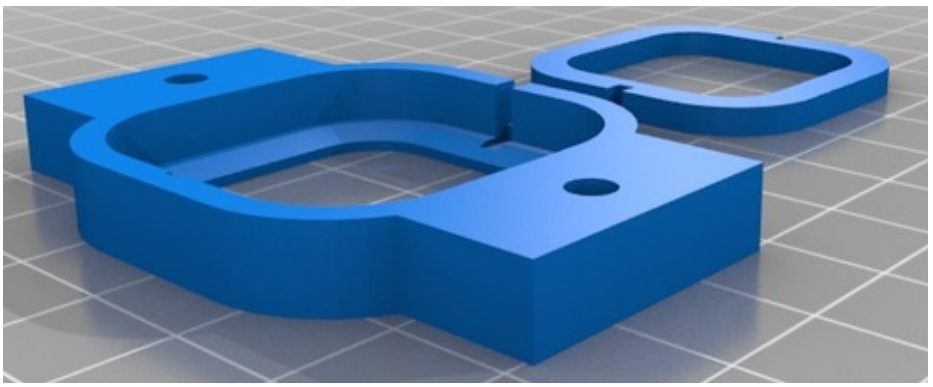


Au centre nous observons un amplificateur d'instrumentation qui est destiné au traitement de signaux électriques faibles. Comme on peut l'observer, cet amplificateur est différentiel.

Pour continuer à m'instruire sur les capteurs de force, après m'être renseigné sur la théorie, je me suis mis à la pratique. Je me suis mis à fabriquer une balance grâce à ce capteur, j'ai donc décidé d'en câbler 4 sur les coins d'une planche pour la stabilité :



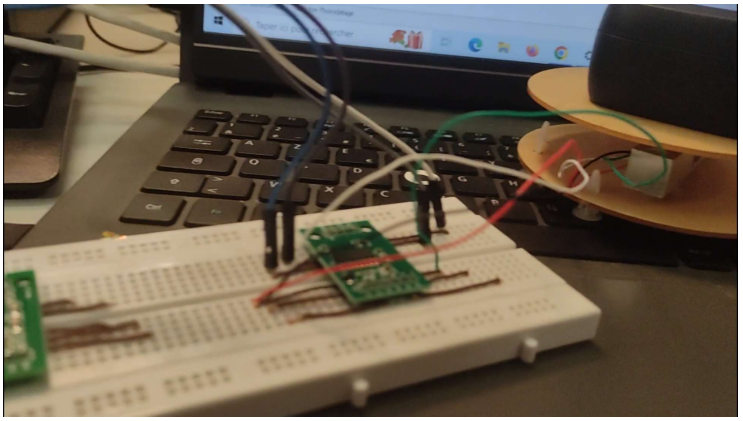
Ces images sont pour la V1, un peu plus tard, je me suis rendu compte qu'il y avait quelques problèmes, que ce n'était pas assez stable et pas assez précis. Notamment car en l'état le capteur manque légèrement de stabilité et induit une erreur de quelques kilos (5-6kg) :



Grâce à ce support, les forces sont mieux réparties entre les capteurs.

Juste après je me suis mis à câbler l'autre capteur :

Balance de cuisine électronique HX711 :



A remplacer par un schéma fritzing

Codes Arduino :

Voici des passages du codes qui permettent la pesée :

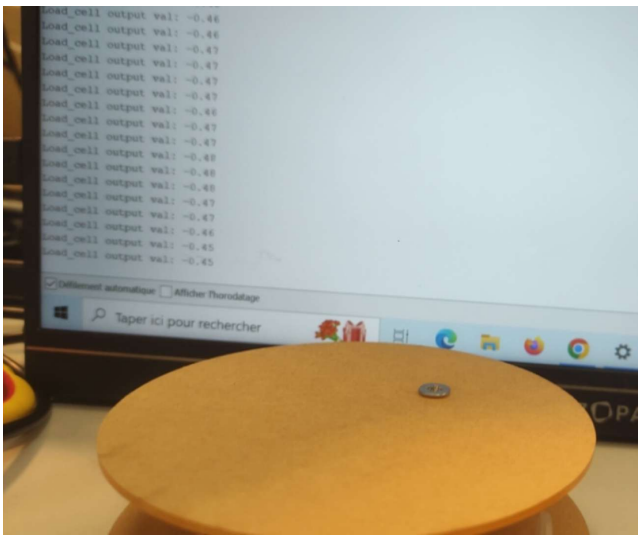
Une partie du code permettant de faire une tare pour augmenter la précision de la pesée

```
// receive command from serial terminal
if (Serial.available() > 0) {
  char inByte = Serial.read();
  if (inByte == 't') LoadCell.tareNoDelay(); //tare
}

// check if last tare operation is complete
if (LoadCell.getTareStatus() == true) {
  Serial.println("Tare complete");
}
```

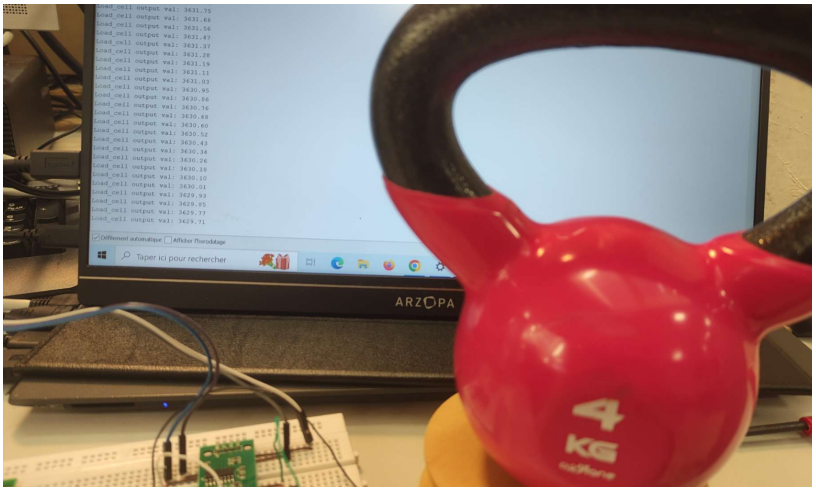
Ici un code lançant la mesure avec la fonction LoadCell.getData();

```
if (newDataReady) {
  if (millis() > t + serialPrintInterval) {
    float i = LoadCell.getData();
    Serial.print("Load_cell output val: ");
    Serial.println(i);
    newDataReady = 0;
    t = millis();
  }
```

Ce capteur a été plus facile d'utilisation et d'installation, je ne pense pas avoir besoin de repasser dessus, les valeurs sont précises et stables :



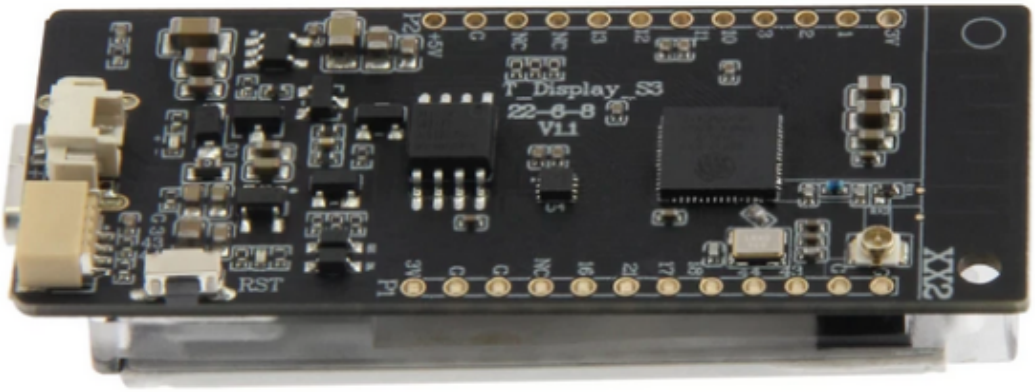
A vide les valeurs données sont très proches de zéro.



Avec un poids de 4kg.

LilyGo :

La carte de développement Lilygo est pourvue d'un écran non tactile de dimension 320x170 en dalle IPS, et du T-display S3 est un ESP32. Sa mémoire flash est de 16MB. Elle peut communiquer en Wifi et en Bluetooth 5.0 ainsi que en SPI, UART, I2C... Elle dispose de 24 entrées/sorties et de 3 boutons poussoirs.

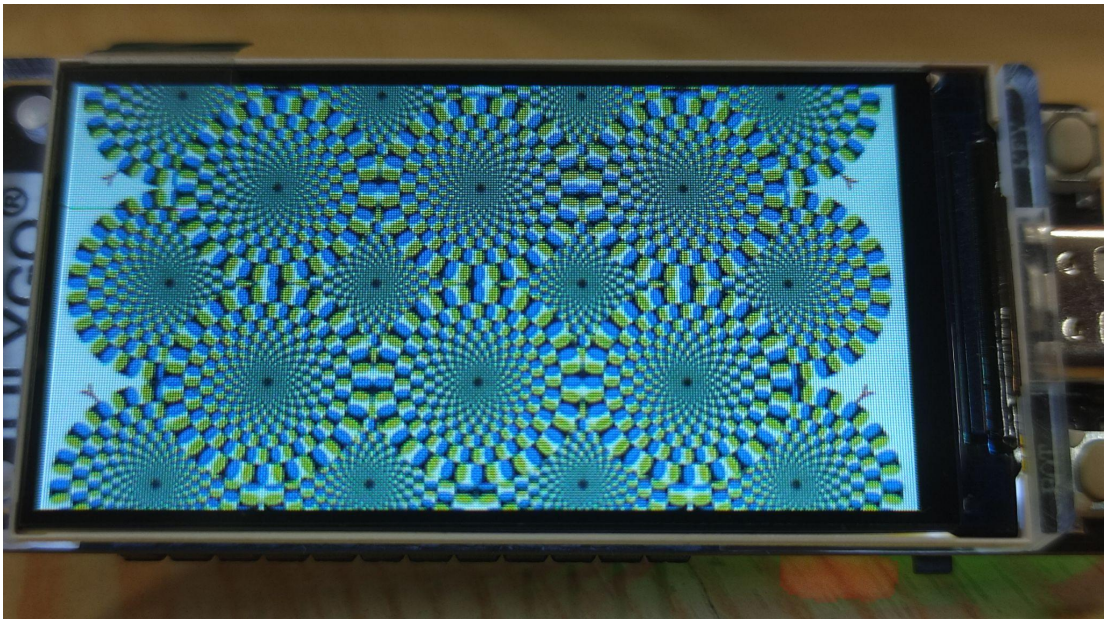


Pour la prise en main de la LilyGo, j'ai appris à afficher une image :

```
#include "TFT_eSPI.h"
#include "dim.h"

TFT_eSPI tft = TFT_eSPI();

void setup() {
  tft.init();
  tft.setRotation(3);
  tft.setSwapBytes(true);
  tft.fillScreen(TFT_BLACK);
  tft.pushImage(0,0,320,170,dim);
}

void loop() {
   tft.setTextDatum(0);
   delay(1000);
}
```

fillScreen pour mettre un fond noir
pushImage pour la placer

dim étant le nom de l'image, pour la rendre compatible avec arduino, il suffit de donner la couleur d'un pixel, pixel par pixel, pour tout l'image, des convertisseurs comme rinkydinkelectronics.com/t_imageconverter565



Ce code génère sur la LilyGo cette image.

Il me suffira de remplacer cette image par une image qui conviendra mieux pour le capteur, d'initialiser une variable et de l'utiliser pour afficher le poids du capteur avec ce code :

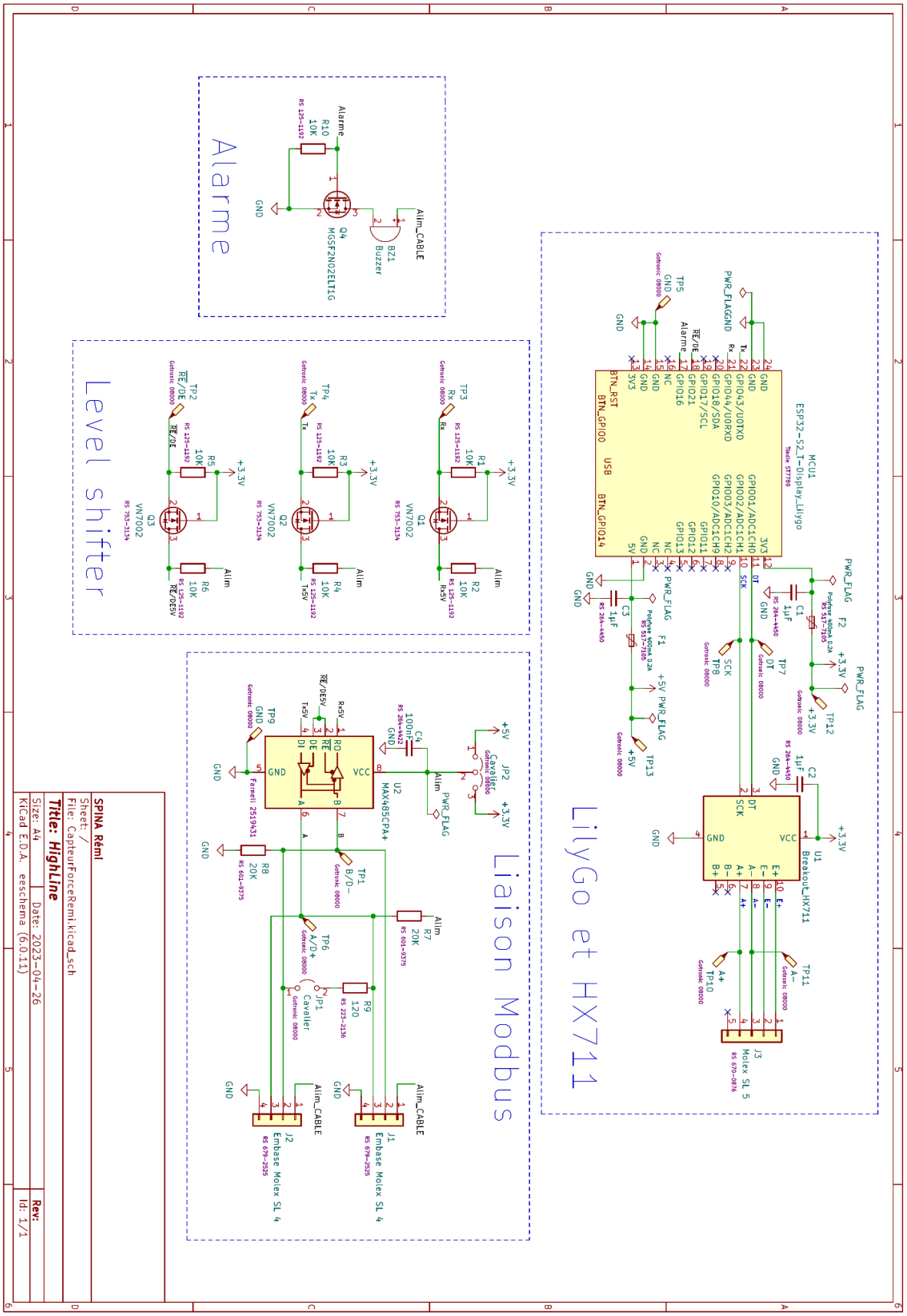
```
#include "Wire.h"
#define DT 17
#define SCK 18

void setup()
{
  Serial.begin(57600);
  Wire.begin(DT, SCK);
}

void loop()
{
  Serial. print ("température en miligrames : ");
  int masse; // temperature in a
  if (Wire.available())
  {
    temperature = Wire.read();
    Serial.print(masse);
    Serial.println("mg");
  }
  else
  {
    Serial.println("---");
  }
  delay(2000);
}
```

Wire.read pour lire les données.

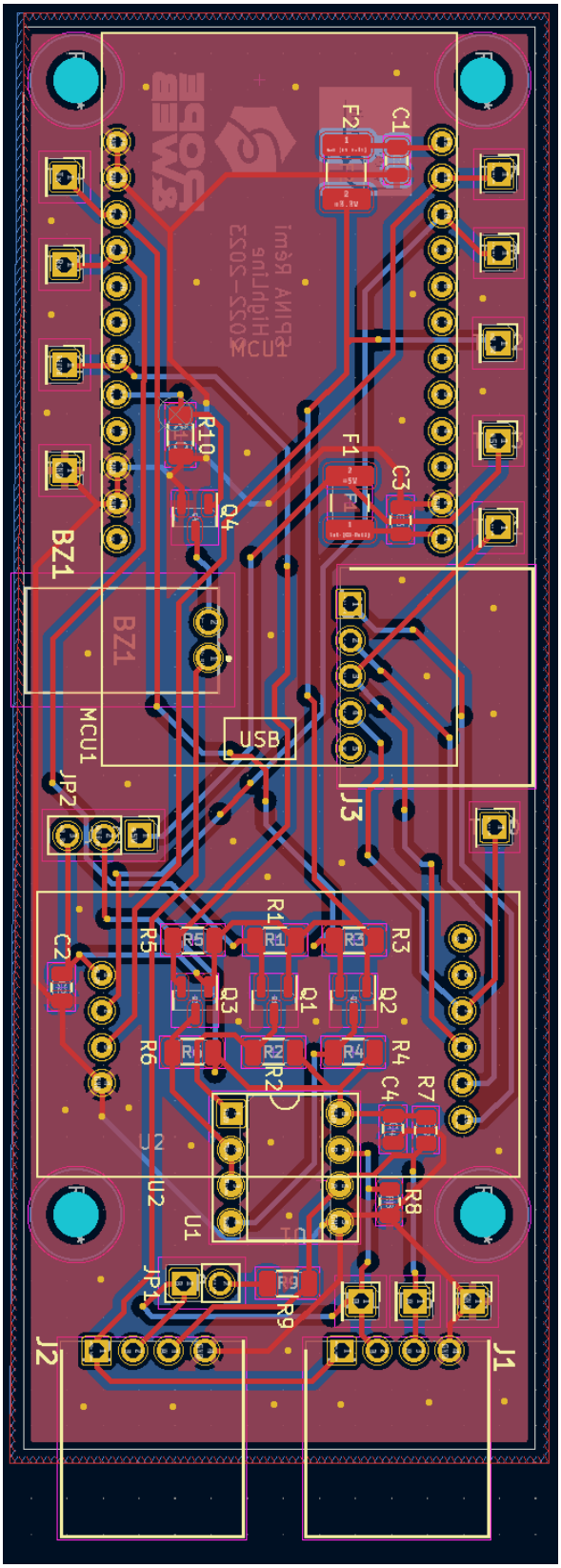
Serial/print pour les afficher.

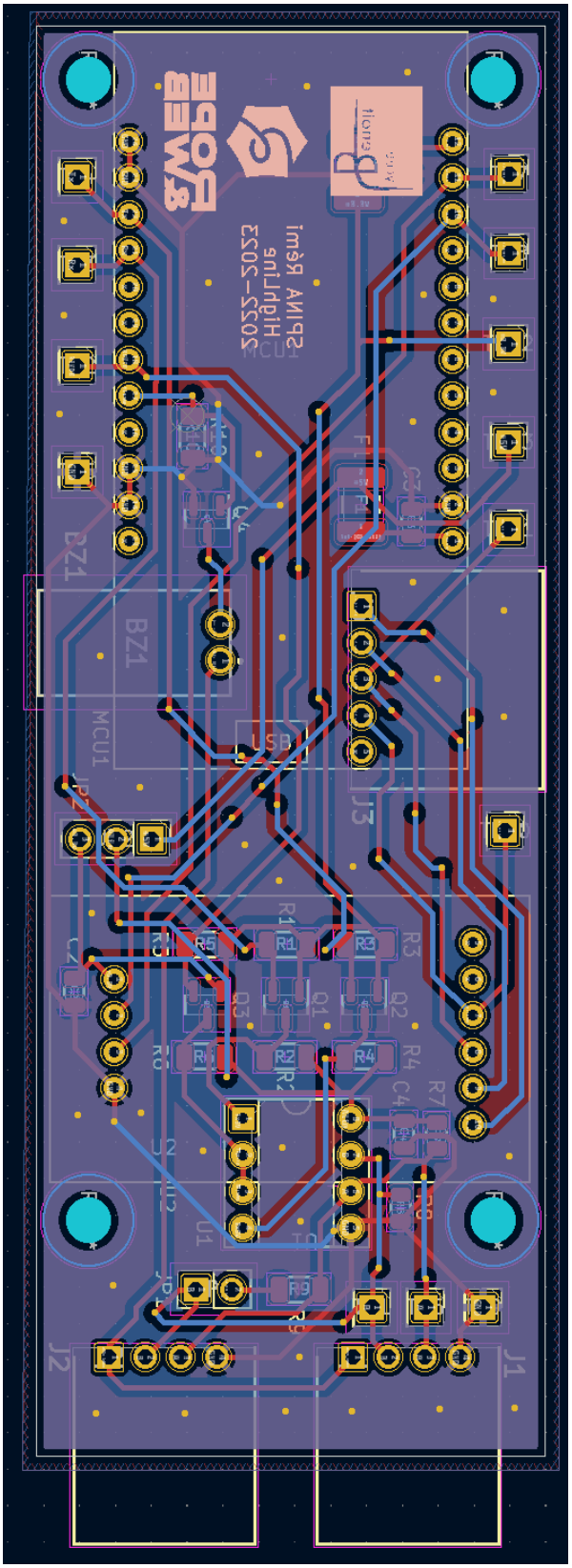
Schéma structurel :

J'ai donc proposé un schéma structurelle d'une carte fille utilisant une transmission ModBus 5V, une adaptation 5V pour les signaux de l'ESP32 étant sous 3.3V ainsi que contenant un module Modbus :

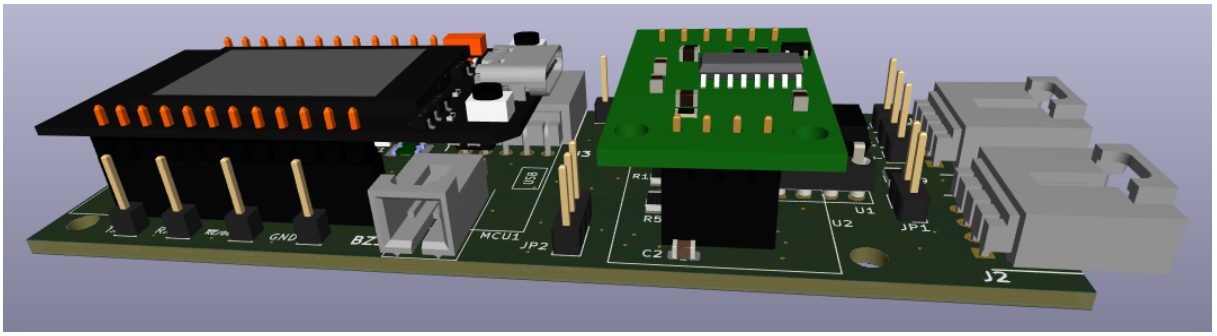


Les faces TOP et BOTTOM :





Vue 3D :



Le fichier BOM de ma carte :

	Référence	Valeur	Code Commande	Quantité
2	BZ1	Buzzer	Gotronic 05460	1
3	> C1-C3	1µF	RS 264-4450	3
4	C4	100nF	RS 264-4422	1
5	> F1, F2	Polyfuse 400mA 0.2A	RS 517-7105	2
6	> J1, J2	Embase Molex SL 4	RS 679-2525	2
7	J3	Molex SL 5	RS 670-0876	1
8	JP1	Cavalier	Gotronic 08000	1
9	JP2	Cavalier	Gotronic 08000	1
10	MCU1	ESP32-S2_T-Display_Lilygo	Tindie ST7789	1
11	> Q1-Q3	VN7002	RS 753-3134	3
12	Q4	MGSF2N02ELT1G	RS 792-5675	1
13	> R7, R8	20K	RS 601-9375	2
14	R9	120	RS 223-2136	1
15	> R1-R6, R10	10K	RS 125-1192	7
16	TP1	B/D-	Gotronic 08000	1
17	TP2	~{RE}/DE	Gotronic 08000	1
18	TP3	Rx	Gotronic 08000	1
19	TP4	Tx	Gotronic 08000	1
20	> TP5, TP9	GND	Gotronic 08000	2
21	TP6	A/D+	Gotronic 08000	1
22	TP7	DT	Gotronic 08000	1
23	TP8	SCK	Gotronic 08000	1
24	TP10	A+	Gotronic 08000	1
25	TP11	A-	Gotronic 08000	1
26	TP12	+3.3V	Gotronic 08000	1
27	TP13	+5V	Gotronic 08000	1
28	U1	Breakout_HX711	Aliexpress	1
29	U2	MAX485CPA+	Farnell 2519431	1

Protocole Modbus :

Modbus est un protocole de communication série créé par Modicon en 1979, il est rapidement devenu un standard dans l'industrie du a son protocole ouvert et sans royauté. Il se base sur le RS 232/485/422. Il existe en 2 versions : RTU (binaire) et ASCII (Hexadécimal). J'utiliserais la version RTU dû à sa praticité et sa sécurité plus importante. Un maître peut commander jusqu'à 247 modules sur un kilomètre et atteint une vitesse de 10Mbaud/s sur une distance courte. La trame maître est une demande composé de :

-L'identification de l'esclave.
-Le code de fonction qui détermine l'action que le maître demande à l'esclave (lire, écrire, …)
-Deux octets d'adresse de départ qui définisse l'adresse du premier registre de l'emplacement mémoire de l'esclave dans lequel le maître veut lire ou écrire.
-Deux octets qui servent à définir le nombre de registres que le maître souhaite lire ou écrire.
-Un code CRC de deux octets pour la détection d'éventuelles erreurs.

La réponse de l'esclave est constitué de

-L'identification de l'esclave.
-Le code de fonction.
-Les données transmises..
-Un code CRC.

Mon but est de communiquer les masses déterminé par le capteur à la Raspberry Pi de M. RICARD Loic.