

EPREUVE E-6-2 POJET TECHNIQUE

MARI-STOVO



COLUS Julia, PINNA Océane, CHASSILLAN Bastien, MARTEL Joey,
CORALLINI Joris

Sommaire

1	Introduction	6
2	Présentation de l'entreprise Mariton	7
3	Objectifs.....	8
4	Partie personnelle (EC1)	9
4.1	Outils et schéma	9
4.1.1	Store utilisé.....	9
4.1.2	Schéma de simulation	10
4.1.3	Composant utilisé	12
4.2	Montage et Tests de la maquette	13
4.2.1	Test avec un seul Relais.....	13
4.2.2	Test avec deux relais	14
4.2.3	Ajout des transistors et de l'Arduino	15
4.3	Programmation	16
4.3.1	Programme Arduino test pour la commande du moteur	16
4.3.2	Programme test pour le capteur ultrason.....	17
4.4	Conception de la carte	18
4.4.1	Schema structurel Isis Proteus final	18
4.4.2	Routage ARES	21
4.4.3	Fichier gerber et conception	22
4.5	Level Shifter (Tuto rédigé pour les membres IR).....	25
	Présentation	26
	Branchement.....	26
	GPIO Raspberry Pi 2	26
	Branchement du Shield	27
	Installation i2c.....	28
4.6	Diagramme SYSML (BDD, IBD et Activités)	30
4.6.1	Diagramme de Block (BDD)	30
4.6.2	Diagramme de Block Interne (IBD)	31
4.6.3	Diagramme d'Activités	32
4.7	Programme final de l'Arduino	33
4.8	Programme Qt (mainwindow.cpp).....	35
4.8.1	Apperçu de l'application Qt.....	36
4.9	Annexes.....	37

4.9.1	Journal de travail :	37
4.9.2	Gantt	39
4.10	Conclusion	41
5	Partie personnelle (IR1)	42
5.1	Présentation du store d'intérieur :	42
5.2	Présentation du matériel :	44
5.2.1	Raspberry + Level Shifter	44
5.2.2	Maquette Vollet	44
5.3	Diagramme UML et Programme :	45
5.4	Extraits du travail réaliser sur le programme :	47
5.5	CONCLUSION	53
5.5.1	Difficultés rencontrées	53
5.5.2	Pour conclure	54
5.6	Annexes	55
5.6.1	Schéma électronique du Store	55
5.6.2	Journal de bord	56
6	Partie personnelle (IR2)	57
6.1	Présentation :	57
6.2	Diagramme de cas d'utilisations :	58
6.3	Diagramme d'exigences :	59
6.4	Diagramme d'état :	60
6.5	Diagramme de séquences :	61
6.6	Programme base de données :	62
6.7	Création d'une base de données test :	64
6.8	Communication I2C :	66
6.9	Programme I2C sous QT-Créator :	68
6.10	Capteur de pluie :	71
6.10.1	Présentation :	71
6.11	Carte Arduino utilisé :	72
6.12	Programme capteur de pluie :	74
6.13	Volet roulant :	76
6.14	Programme sous QT-Creator pour le volet roulant :	78
6.15	Notion de physique :	80
6.15.1	I2C :	80

6.16	Partie : Capteur de pluie :	82
6.17	Conclusion.....	83
6.17.1	Ressentie du projet :	83
6.17.2	Difficultés rencontré :	83
6.17.3	Pour le moment :	84
6.18	Annexes programme .cpp du volet roulant :	85
7	Partie personnelle (IR3).....	88
7.1	Présentation de la Raspberry PI	89
7.2	Diagrammes	90
7.3	Programme	94
7.3.1	Création de la base de donnée pour permettre de vérifié le bon fonctionnement du programme :	94
7.3.2	.h : fichier d'entête.....	95
7.3.3	main.cpp.....	96
7.3.5	Interface Graphique	99
7.4	Partie physique :	100
7.5	Problèmes rencontrés	101
7.7	Remerciements	101
8	Partie personnelle (IR4).....	102
8.1	Présentation du Projet:.....	102
8.1.1	Mon Rôle dans le projet:	102
8.2	Diagrammes:	103
8.3	Environnement de Travail:.....	104
8.4	Application TCP:	105
8.4.1	Le Serveur TCP:	105
8.4.2	Le Client TCP:.....	105
8.4.3	Diagrammes:.....	106
8.5		106
8.5.1	Communication Client/serveur TCP:.....	107
8.5.2	QTCPSEVEUR :	107
8.6	Base de Données:	108
8.6.1	Communication BDD:.....	109
8.7	Le Site Web:	110
8.7.1	Langages de Programmation:.....	110

8.7.2	Principe de Fonctionnement:	111
8.8	Les Capteurs :	116
8.8.1	Capteur de luminosité :	116
8.8.2	Capteur de pluie :	116
8.8.3	Capteur de Vent :	116

1 Introduction

Le projet MARI-STOVO est un projet avec pour objectif de prolonger les activités de la société dans le domaine de la domotique et d'acquérir une autonomie dans la fourniture de matériels et logiciels nécessaires pour l'intégration aux produits existants.

Notre groupe (COLUS Julia, PINNA Océane, CHASSILIAN Bastien, MARTEL Joey, CORALLINI Joris) se prépare à passer le brevet de technicien supérieur. Pour cela il nous a été demandé de gérer les stores d'intérieur et d'extérieurs ainsi que les volets roulants dans un but de confort, de sécurité et d'économie d'énergie pour l'entreprise Mariton.

2 Présentation de l'entreprise Mariton

L'entreprise a été créée en 1949 par Jean Mariton, menuisier et ébéniste, à Saint Chamas au 31 boulevard Joliot curie 13250. En 1972 L'entreprise fût immatriculée avec son numéro de SIRET : 637 280 272 00013.

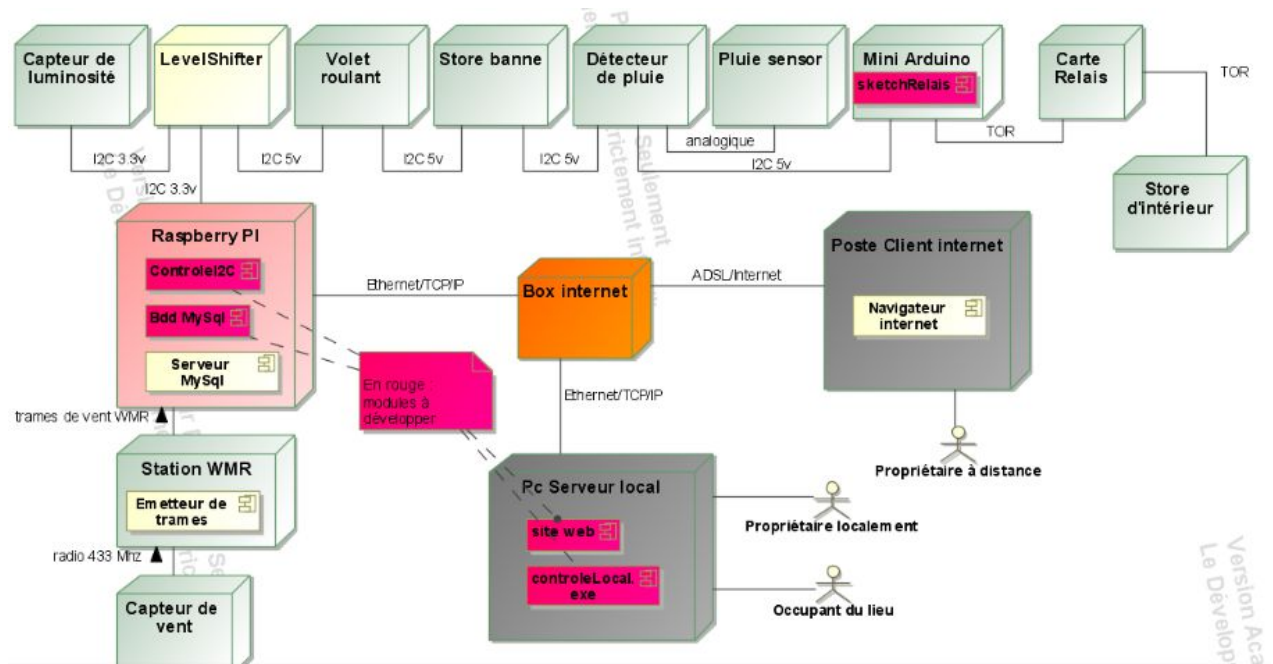
Dirigé aujourd'hui par Yves Mariton, cette société (PME) est actuellement le leader de son secteur d'activité en France. Mariton est spécialisé dans la fabrication de stores et de moustiquaires sur mesure pour portes, et fenêtres. Depuis plus de 60 ans Mariton s'occupe aussi bien de l'équipement des maisons que des entreprises, collectivités, etc.



Exemple de store enrouleur

Cette société fabrique des stores enrouleurs, des stores à bandes verticales, des stores vénitiens en aluminium ou en bois, des stores plissés, stores bateau, ou encore des panneaux japonais.

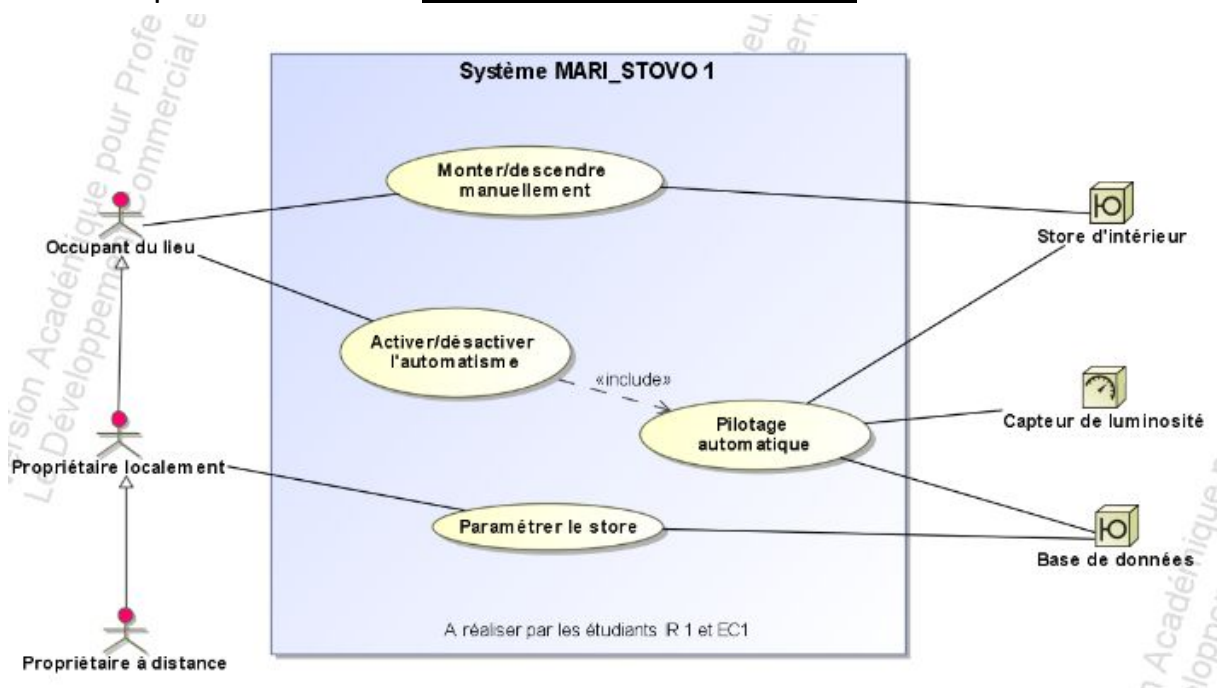
3 Objectifs



Ci-dessus le diagramme de block montrant l'ensemble du projet.

Nous avons donc à notre disposition plusieurs types de stores et de capteurs ayant chacun leur fonction. L'un pour cacher la lumière du jour, un autre pour protéger de la pluie, etc.

Pour chaque store il y a deux modes de fonctionnement: Manuel ou automatique. Ci-dessous le Diagramme cas d'utilisation:



4 Partie personnelle (EC1)

CORALLINI Joris

Le but principal de ma partie est de pouvoir commander un store manuellement que ce soit à distance ou grâce à une commande murale.

4.1 Outils et schéma

4.1.1 Store utilisé

Pour ma partie j'effectue mes tests sur un store plissé de la marque Somfy. Il peut se dérouler sur 1 mètre 80 centimètre.

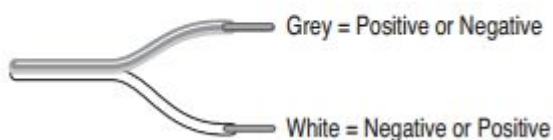
Le moteur utilisé pour ce store est un moteur Somfy LW25-B44.

CHARACTERISTICS					
Supply voltage	24 V DC				
Degree of protection	IP 20				
Operating temperature	-10 to 60°C for LV25 B64 , LW25 B44 and LW25 B83 -10°C to 85°C for LW25 E83				
Protection class	Class III				
Limit switch unit	None				
Conformity	www.somfy.com/ce				

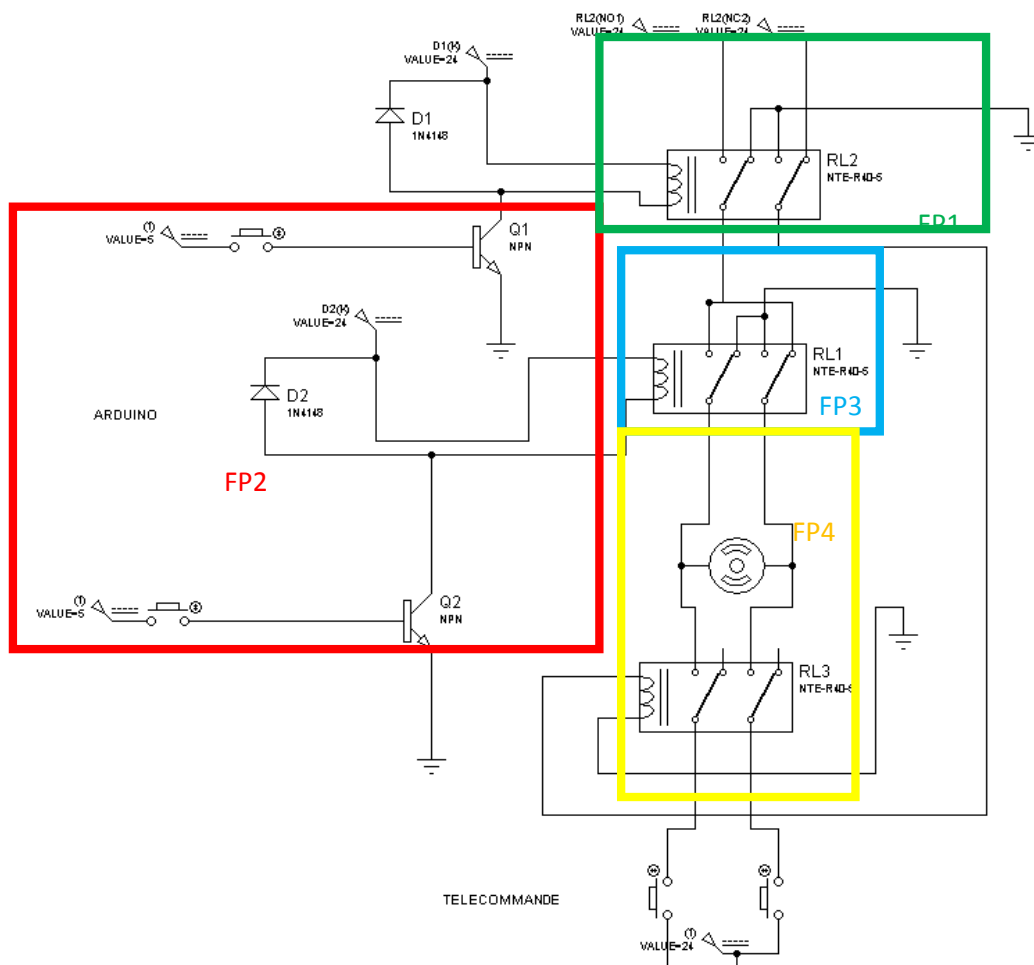
Performance					
	Single put motor		Double output motor		
	LV25 - B44	LV25 - B64	LW25 - B44	LW25 - B83	LW25 - E83
Nominal voltage	24 V DC		24 V DC		
Torque	0,4 Nm	0,6 Nm	0,4 Nm	0,8 Nm	0,8 Nm
Speed	40 rpm	35 rpm	40 rpm	30 rpm	30 rpm

Ce moteur est composé de deux fils pouvant être chargé positivement ou négativement :

LV25 and LW25
2 threads by wire



4.1.2 Schéma de simulation



Ceci est un schéma simplifié du montage de test :

FP1 : Choisir entre commande Arduino ou murale.

FP2 : Commande Arduino.

FP3 : Sens moteur (Arduino).

FP4 : Sens moteur (commande murale).

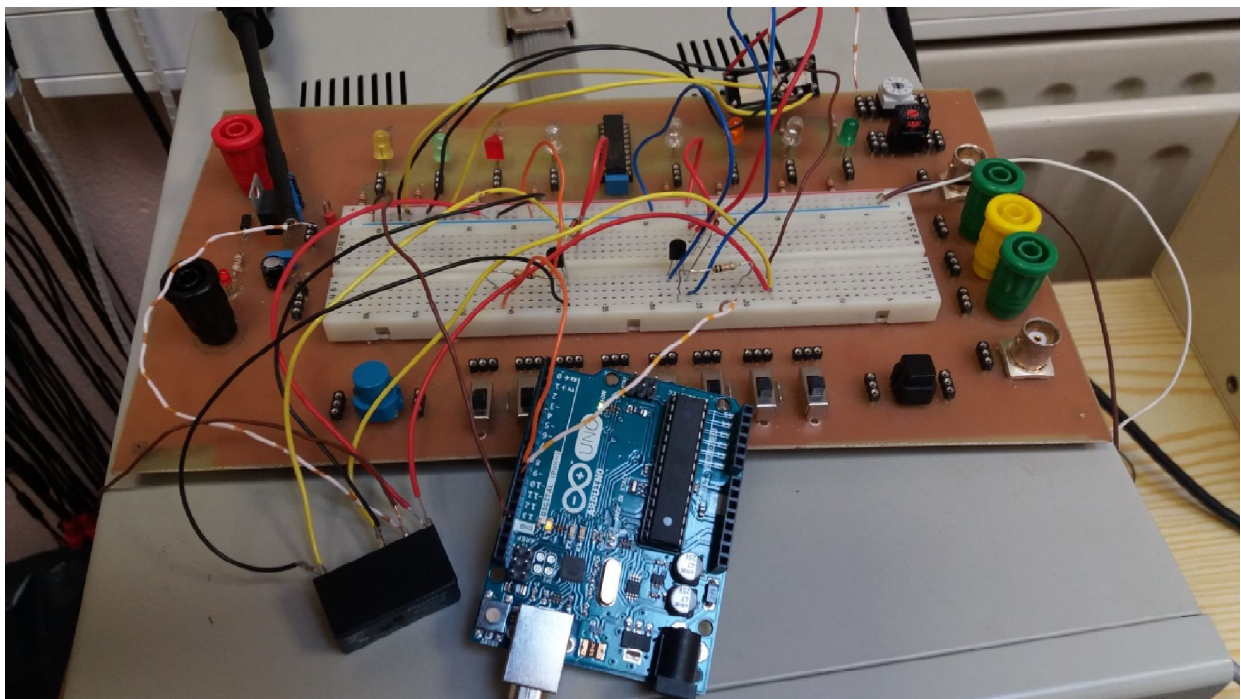
Trois relais sont utilisés pour la commande moteur. Un premier permettant de choisir entre une commande par télécommande murale ou par Arduino.

Un deuxième pour changer le sens de dépliement du store avec l'Arduino. Et le troisième pour commander le store avec la commande murale.

Deux Pin digital de l'Arduino sont utilisés pour piloter le store :

-Pin 10 → si l'Arduino reçoit un « 1 » de part le moniteur série alors le relais est alimenté et le moteur tourne de sorte à déplier le store, inversement si elle reçoit un « 0 » alors ce sera à la commande murale de commander le moteur.

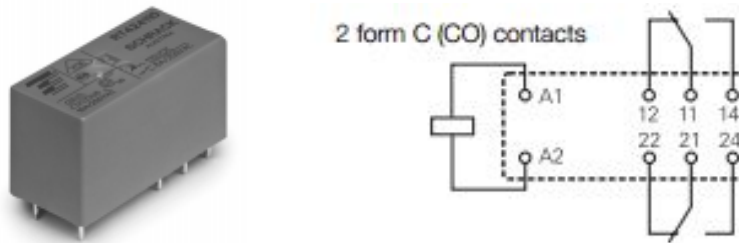
-Pin 9 → si l'Arduino reçoit un « 2 » alors le moteur change de sens grâce au relais qui change la polarisation du moteur de telle sorte à changer son sens de rotation. (Ceci est seulement pour la phase de test. Les commande se feront par i2c à la fin du projet).



Maquette test

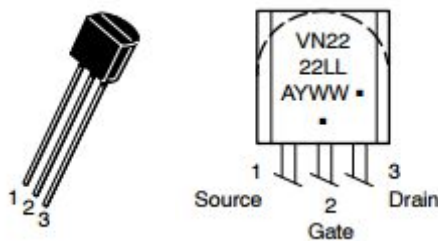
4.1.3 Composant utilisé

Les relais utilisés pour mes tests sont les relais RTE24024F de TE Connectivity.



Il suffit de 24 Volts à la bobine pour actionner le relais. La tension que peut faire passer ce relais est de 24Volts.

Pour activer les Pin digitale de l'Arduino j'utilise des transistors MOSFET VN2222LLG canal N.



Rating	Symbol	Value	Unit
Drain – Source Voltage	V_{DSS}	60	Vdc
Drain–Gate Voltage ($R_{GS} = 1.0 \text{ M}\Omega$)	V_{DGR}	60	Vdc
Gate–Source Voltage	V_{GS}	± 20	Vdc
– Continuous	V_{GSM}	± 40	Vpk
– Non–repetitive ($t_p \leq 50 \mu s$)			
Drain Current	I_D	150	mAdc
– Continuous	I_{DM}	1000	
– Pulsed			
Total Power Dissipation @ $T_A = 25^\circ\text{C}$	P_D	400	mW
Derate above 25°C		3.2	mW/ $^\circ\text{C}$
Operating and Storage Temperature Range	T_J, T_{stg}	–55 to +150	$^\circ\text{C}$

Au départ de ce projet j'avais prévu de calculer la distance du store avec un retour d'information de ce dernier grâce à un troisième fil. Malheureusement sans succès. (Pour cause d'incompatibilité avec du matériel autre que constructeur).

Pour calculer la distance de dépliement du store j'utilise donc un MB1030 LV-MaxSonar-EZ3. C'est un capteur ultrason (à la fois émetteur et récepteur).



Alimenté en 5 Volt il peut détecter un obstacle jusqu'à 600 centimètres soit 6 mètres. Il a un angle de détection d'environ 20°. Il a une précision de 1 pouce soit 2.54 centimètres.

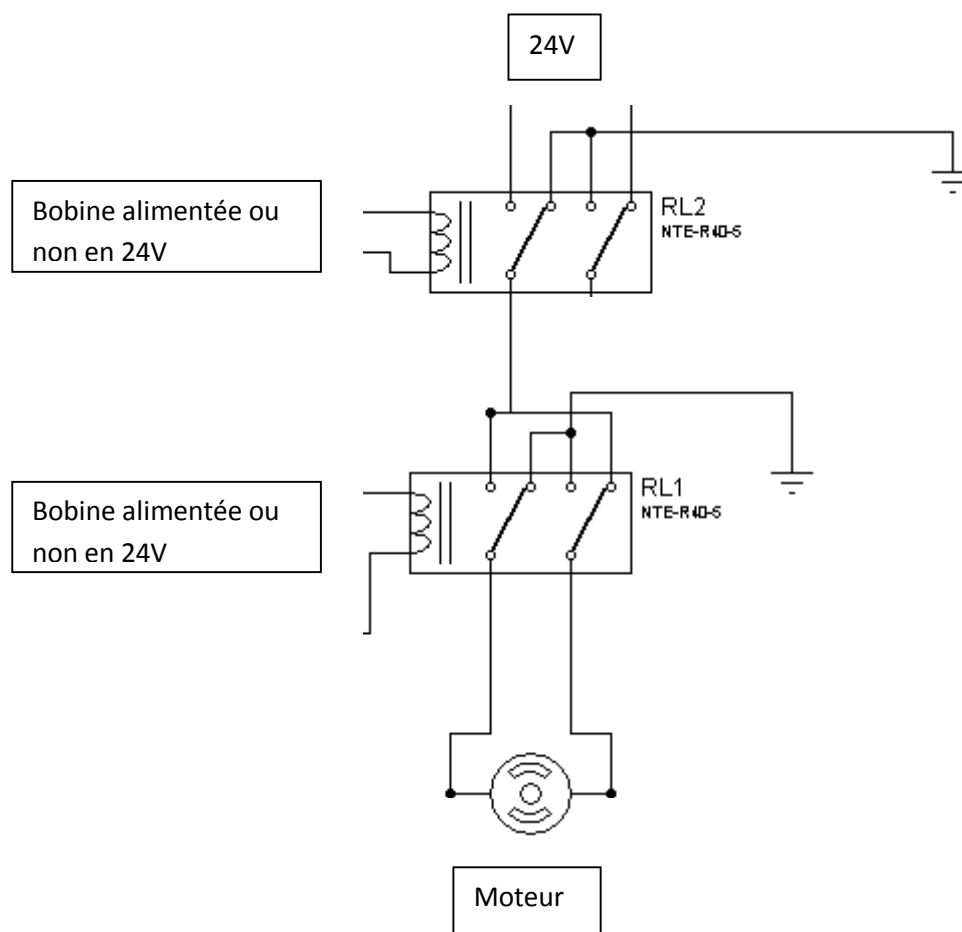
4.2 Montage et Tests de la maquette

4.2.1 Test avec un seul Relais

J'ai d'abord essayé de faire marcher le store avec un seul relais et un générateur de 24 Volts. Le store pouvait changer de sens lorsque le relais commutait ou non. (Relais non alimenté = descente du store / Relais alimenté = montée du store)

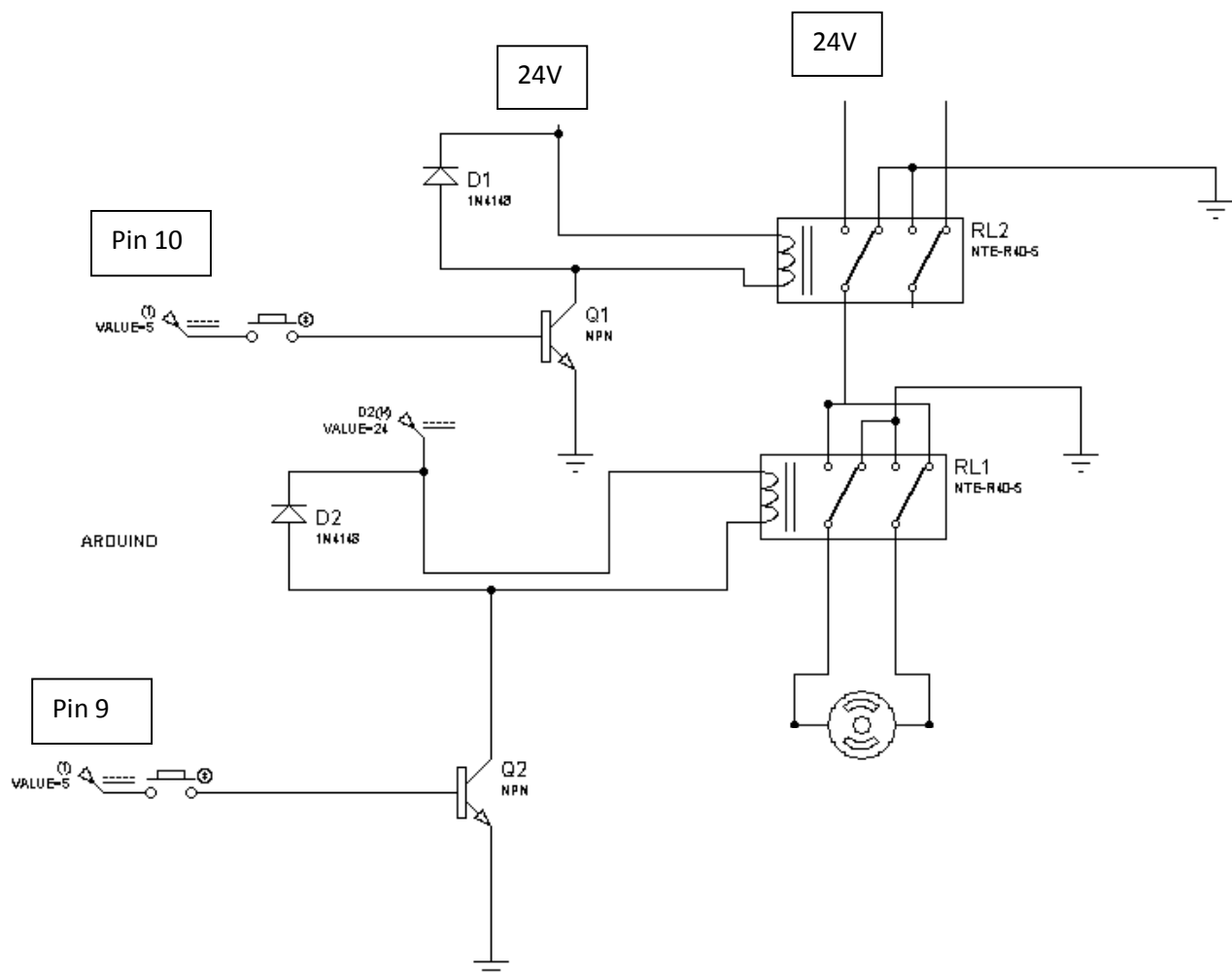
4.2.2 Test avec deux relais

Puis j'ai ajouté un second relais qui fonctionnera comme un « interrupteur » ON/OFF. Grâce à ça on peut changer le sens de rotation même lorsque le store n'est pas sous tension.



4.2.3 Ajout des transistors et de l'Arduino

Enfin j'ajoute une Arduino pour ensuite créer un programme test. Si la pin 10 est à l'état haut le premier relais s'actionne et le moteur se met en marche, à l'inverse à l'état bas le moteur s'arrête. La pin 9 à l'état haut actionne le deuxième relais qui change le sens du moteur.



Ce schéma est toujours une simulation (les broches de l'Arduino sont représentées par des interrupteurs).

4.3 Programmation

4.3.1 Programme Arduino test pour la commande du moteur

```

const int Moteur = 10;
const int Sens = 9;

void setup()
{
  pinMode(Moteur, OUTPUT);
  pinMode(Sens, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available() > 0)
  {
    Recup = Serial.read() ;      //si l'Arduino reçoit 1 alors le moteur marche
    if (Recup == '1')
    {
      digitalWrite(Moteur, HIGH);
      delay (1000);

    }
    else
    {
      //rien
    }
    if (Recup == '0')      //si l'Arduino reçoit 0 alors le moteur s'arrête
    {
      digitalWrite(Moteur, LOW);
    }
    else
    {

    }

    if (Recup == '2')  //si l'Arduino reçoit 2 le store monte
    {

      digitalWrite(Sens, HIGH);
    }
    else
    {

    }

    if (Recup == '3')      //si l'Arduino reçoit 3 le store descend
    {

      digitalWrite(Sens, LOW);
    }
    else
    {

    }
  }
}

```

```

        Serial.print("I received: ");
        Serial.println(Recup);
    }
}

```

4.3.2 Programme test pour le capteur ultrason

```

#include <Maxbotix.h>

long anVolt, inches, cm;
int sum=0;//Create sum variable so it can be averaged
int avgrange=60;//Quantity of values to average (sample size)
Maxbotix rangeSensorAD(A1, Maxbotix::AN, Maxbotix::LV, Maxbotix::BEST, 9);

char Recup ;

void setup()
{
    Serial.begin(9600);
    rangeSensorAD.setADSampleDelay(10);
}

void loop()
{
    unsigned long start;

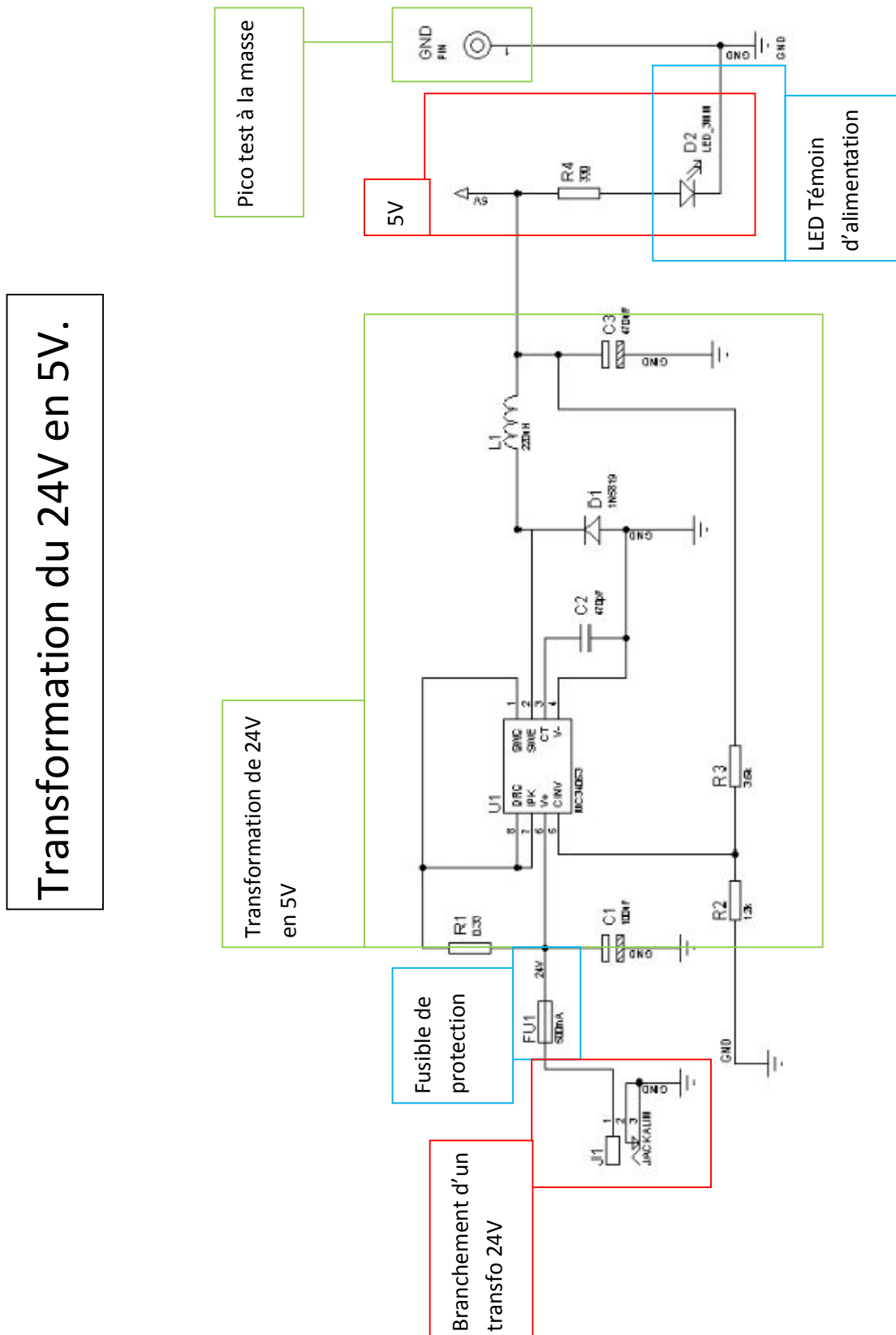
    //AD
    Serial.println("Reading...");
    start = millis();
    Serial.print("AD (BEST): ");
    Serial.print(rangeSensorAD.getRange());
    if (rangeSensorAD.getRange() < 182)
    {
        Serial.print("cm - ");
        Serial.print(millis() - start);
        Serial.print("ms - ");
        Serial.print(" - Highest Mode: ");
        Serial.print(rangeSensorAD.getSampleMode(true));
        Serial.print(" - Lowest Mode: ");
        Serial.print(rangeSensorAD.getSampleMode(false));
        Serial.print(" - Median: ");
        Serial.print(rangeSensorAD.getSampleMedian());
        Serial.print(" - Best: ");
        Serial.print(rangeSensorAD.getSampleBest());
        Serial.println();

        Serial.println();
        delay(500);
    }
    else{}
}

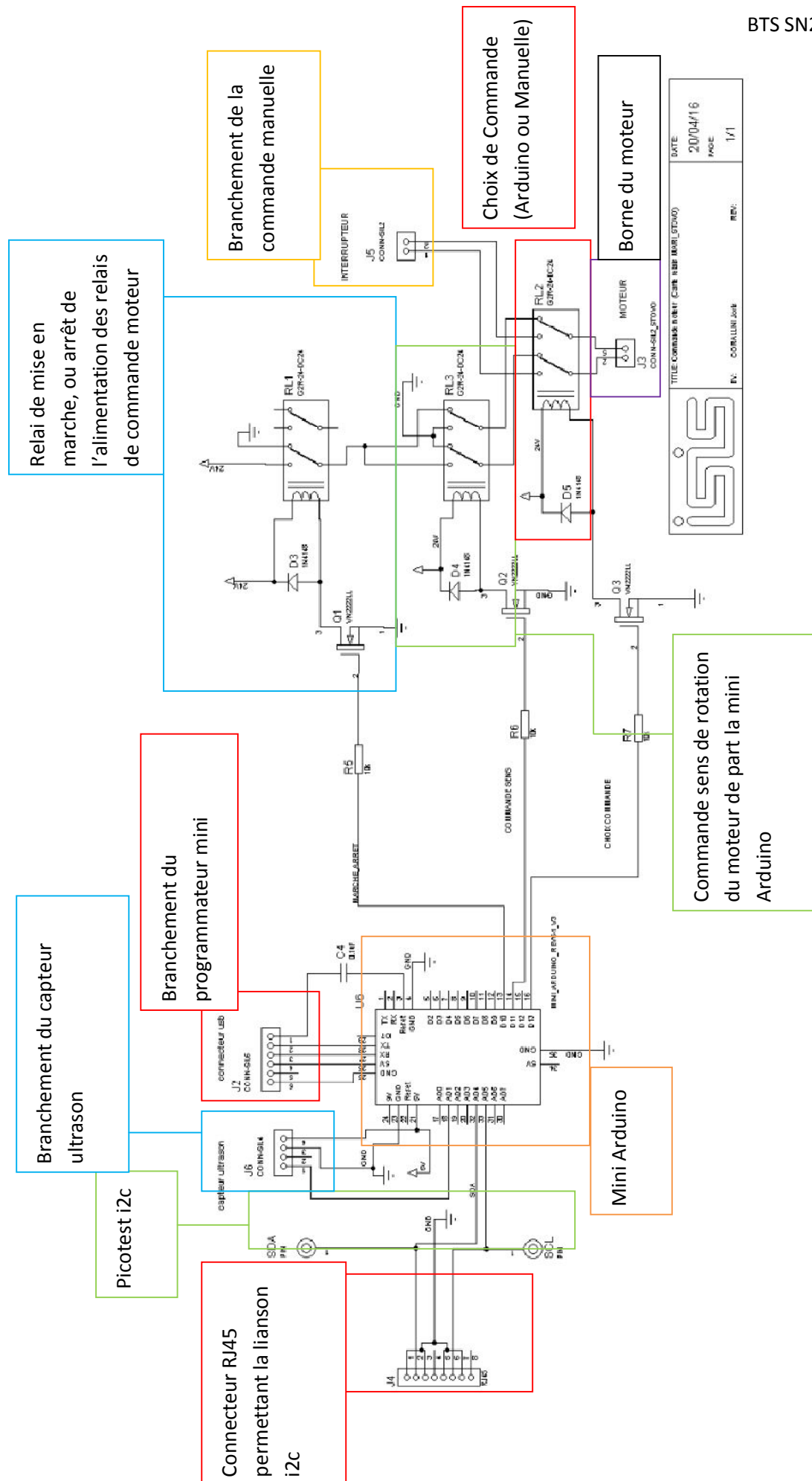
```

4.4 Concetpion de la carte

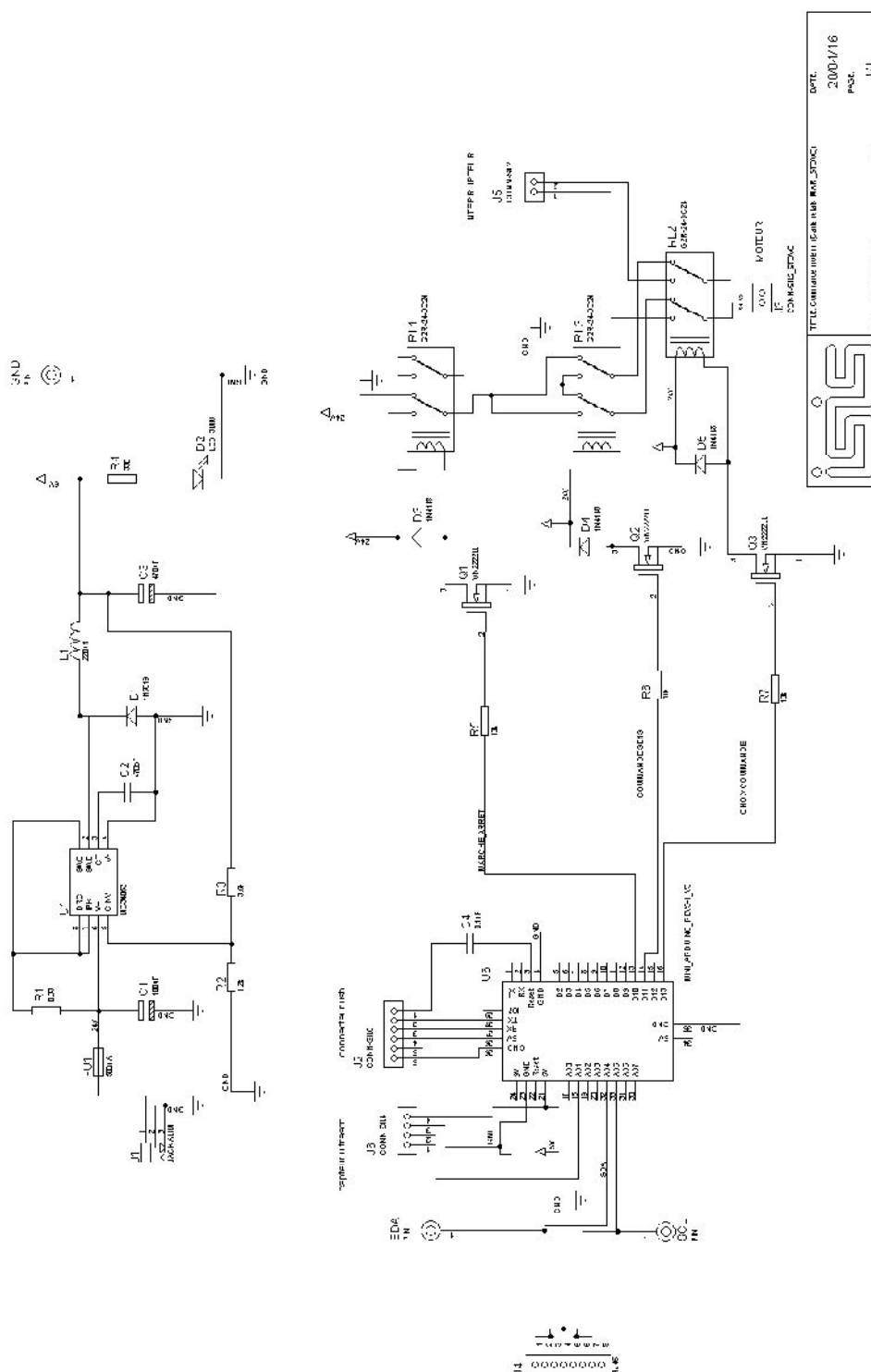
4.4.1 Schema structurel Isis Proteus final



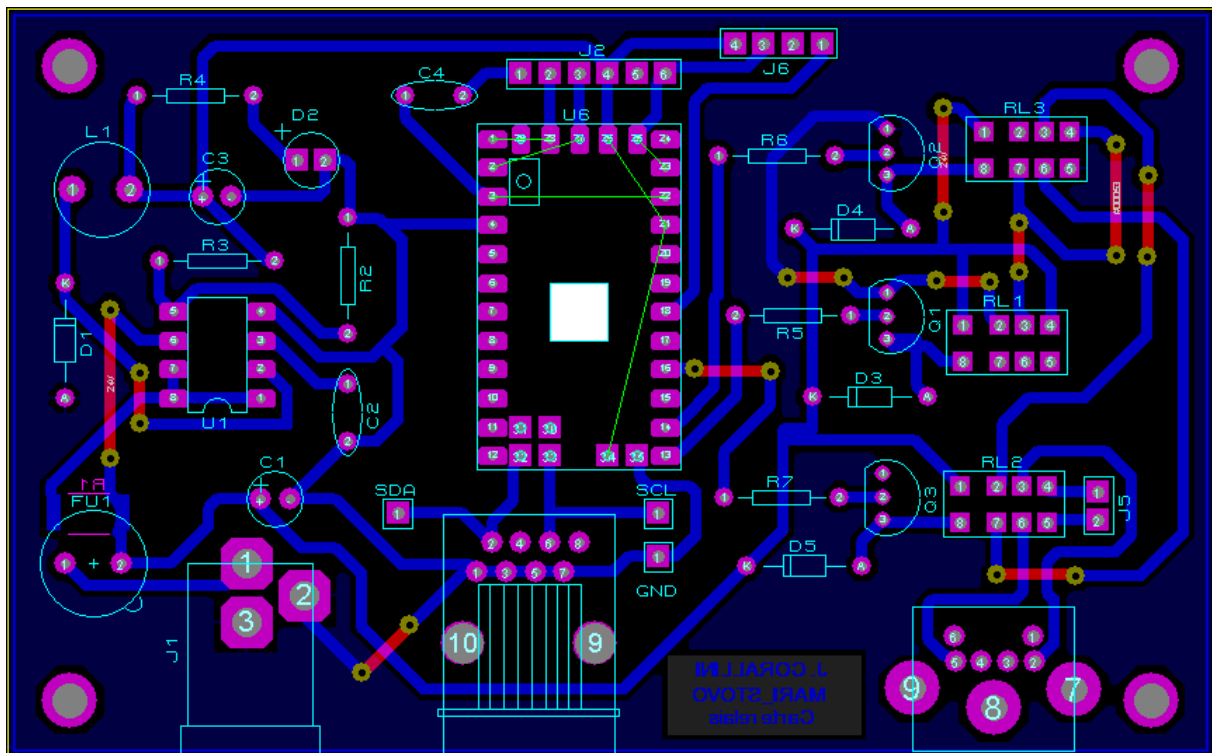
COMMANDE MOTEUR



Schema structurel entier :

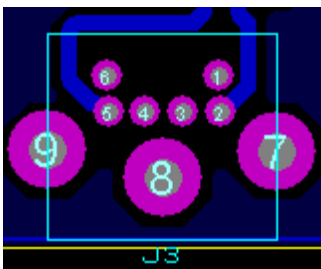


4.4.2 Routage ARES



Voici ci-dessus le routage ARES servant à la conception de la carte :
 Sur la gauche on peut voir la partie du circuit servant à transformer le 24V en 5V. Au milieu la mini Arduino avec en dessous le RJ45 et au dessus le connecteur USB servant à la programmation de la mini Arduino.

Sur la partie droite on peut voir les 3 transistors avec les 3 relais. Les relais que j'utiliserai pour la carte sont les AXICOM IMO7N 24DC. Ils fonctionnent de la même manière que mes relais de test.

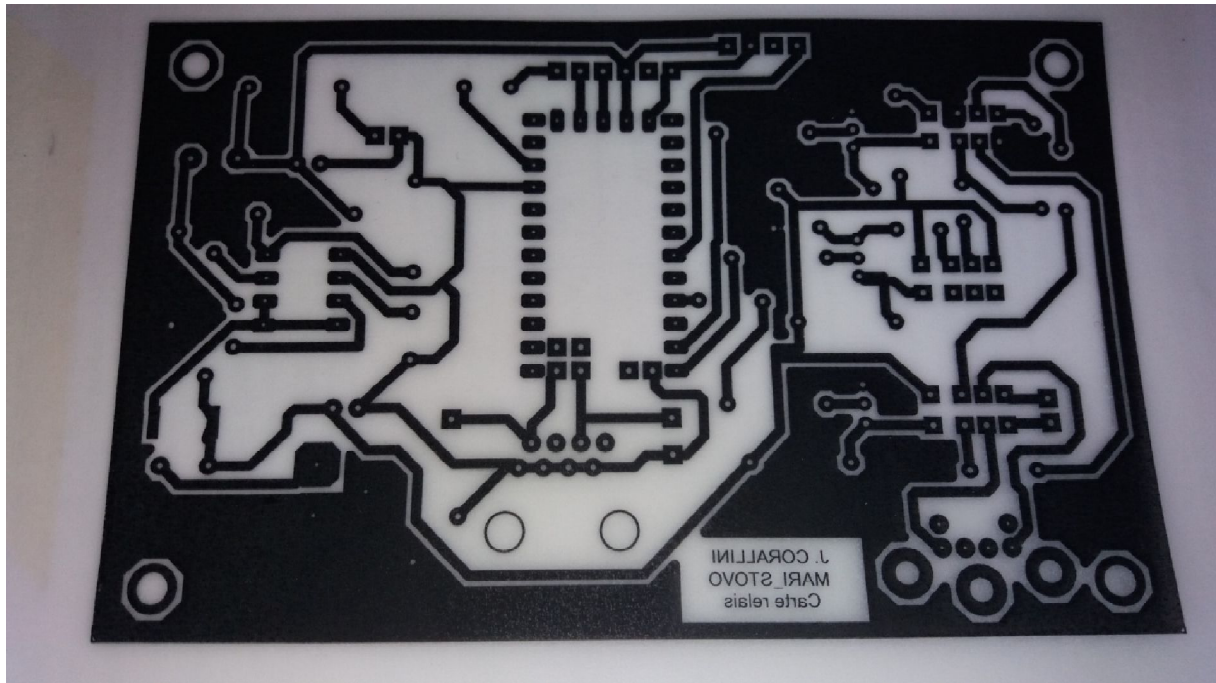


Ceci est un connecteur femelle DIN 6 broches servant à brancher le moteur du store. Le connecteur DIN mâle est déjà installé sur le store de part le projet Mariton précédent.

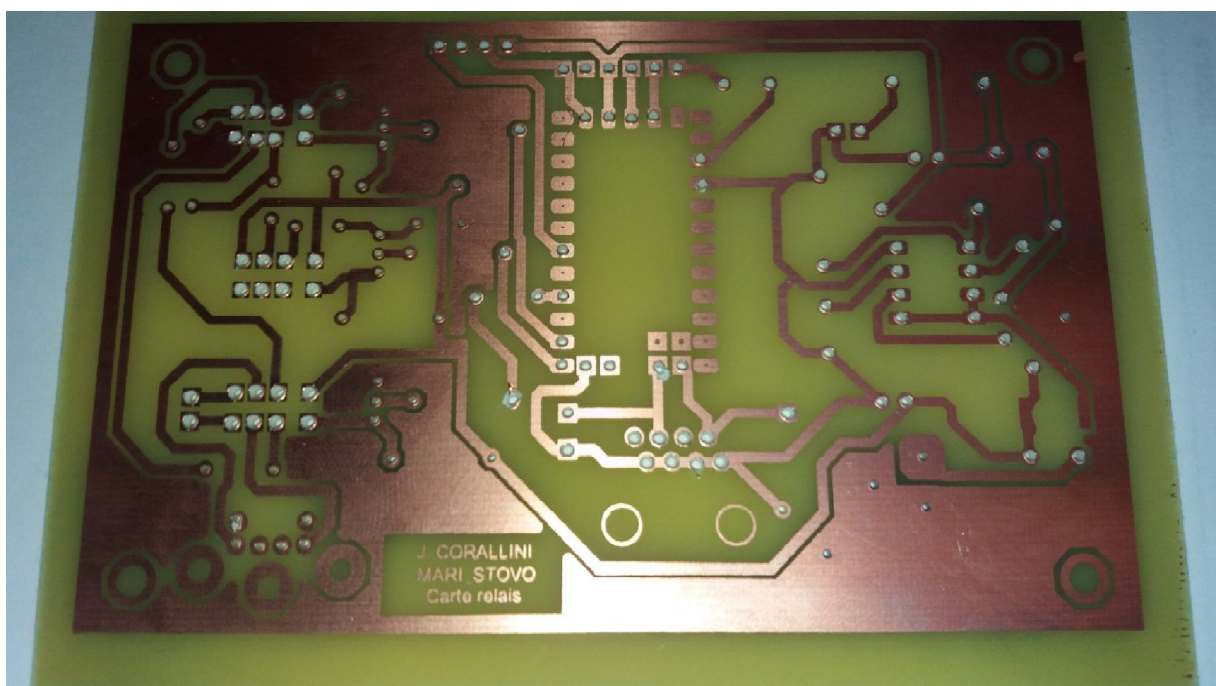
Je ne me sers pas des 6 broches mais seulement de 2, car le moteur à seulement besoin de deux fils.

4.4.3 Fichier gerber et conception

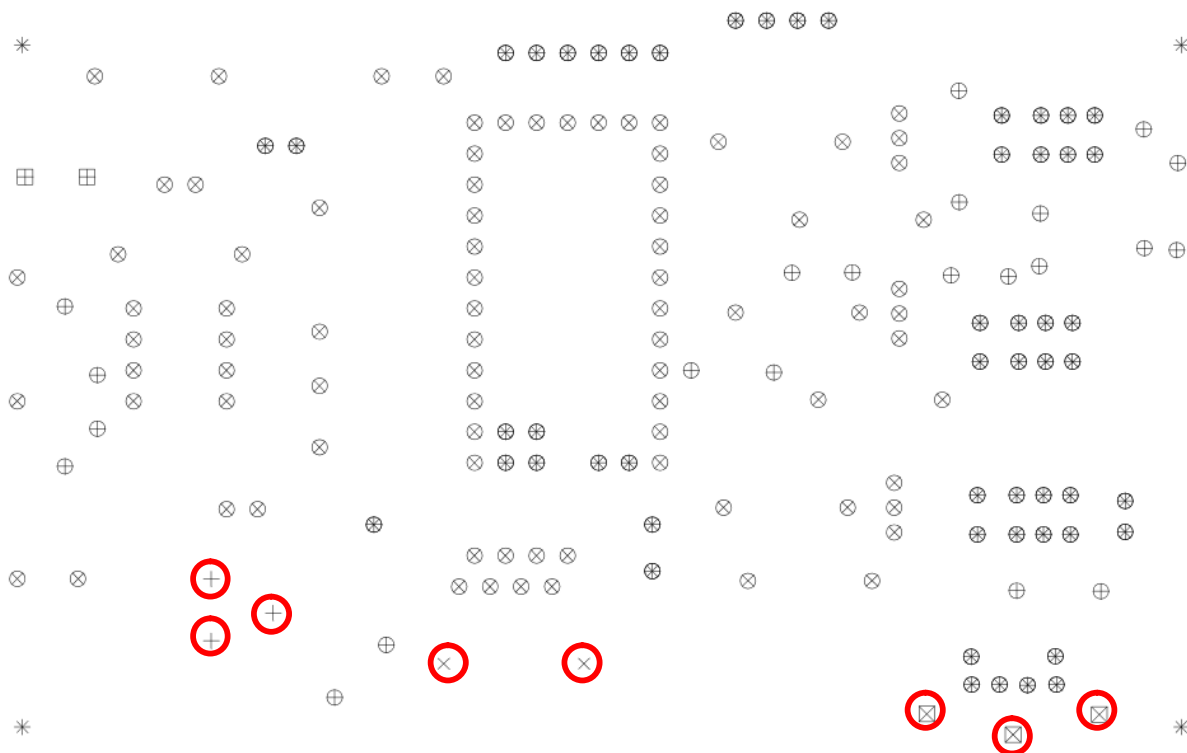
Fichier GERBER imprimé pour la gravure de la carte :



Résultat après la gravure :



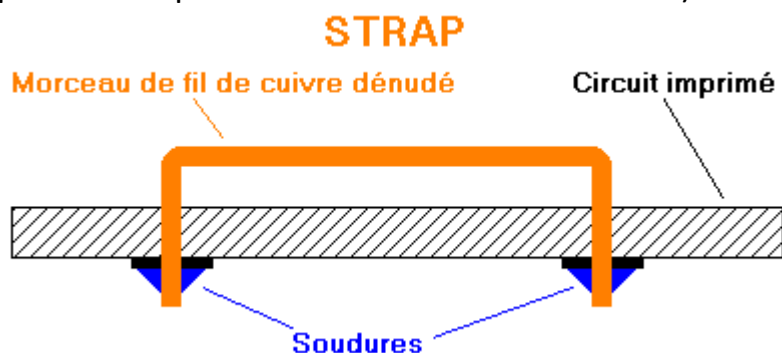
Trou de perçage de ma carte :



Tous les trous sont percés en 1 mm sauf les trous du Jack d'alimentation, les deux trous de la prise RJ45 et les 3 trous du DIN 6.

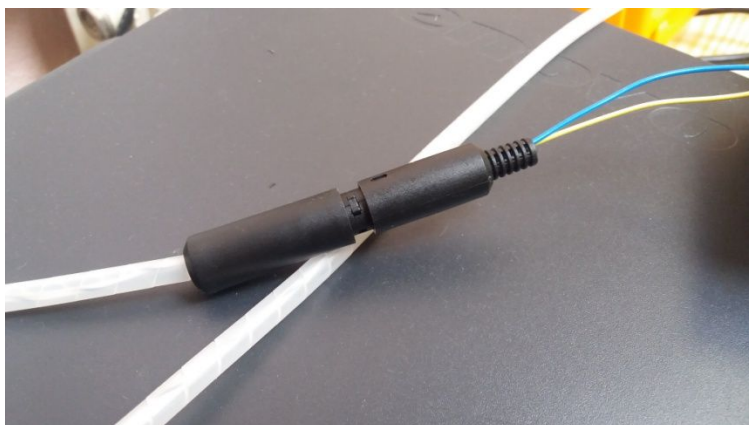
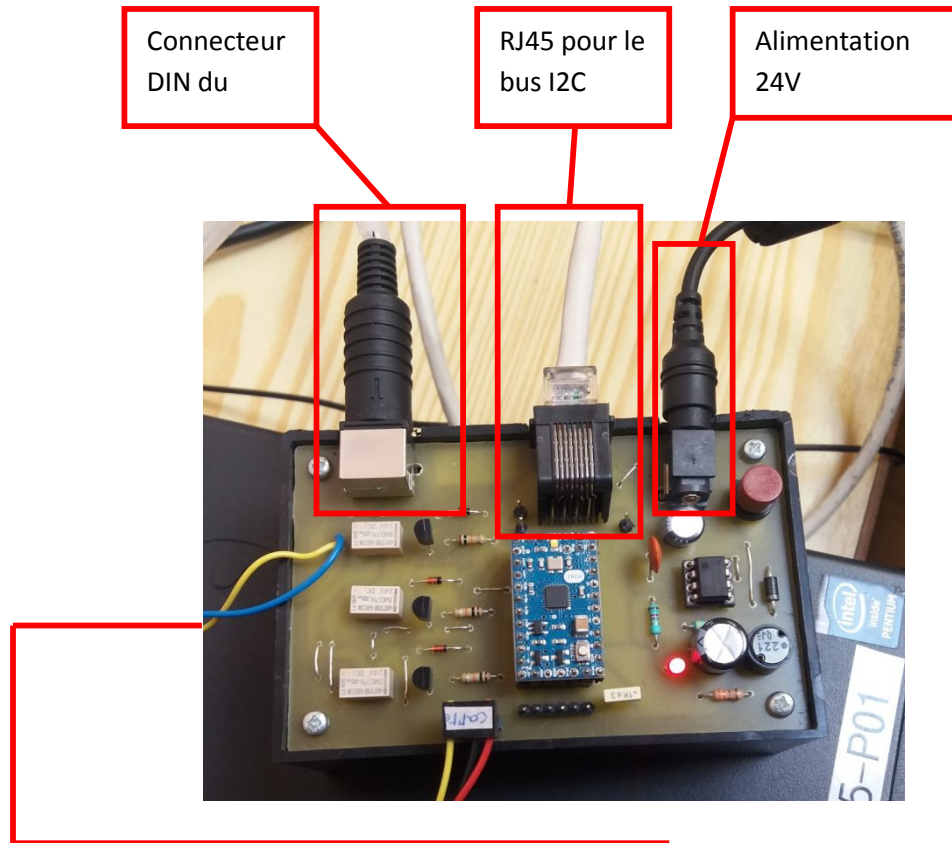
Pour rendre la soudure de mes composants plus facile, j'ai mis en place un ordre de soudure du plus petit composant au plus gros :

-J'ai donc commencé par les straps (fonctionne comme un pont pour faire passer mes pistes de l'autre côté de ma carte).



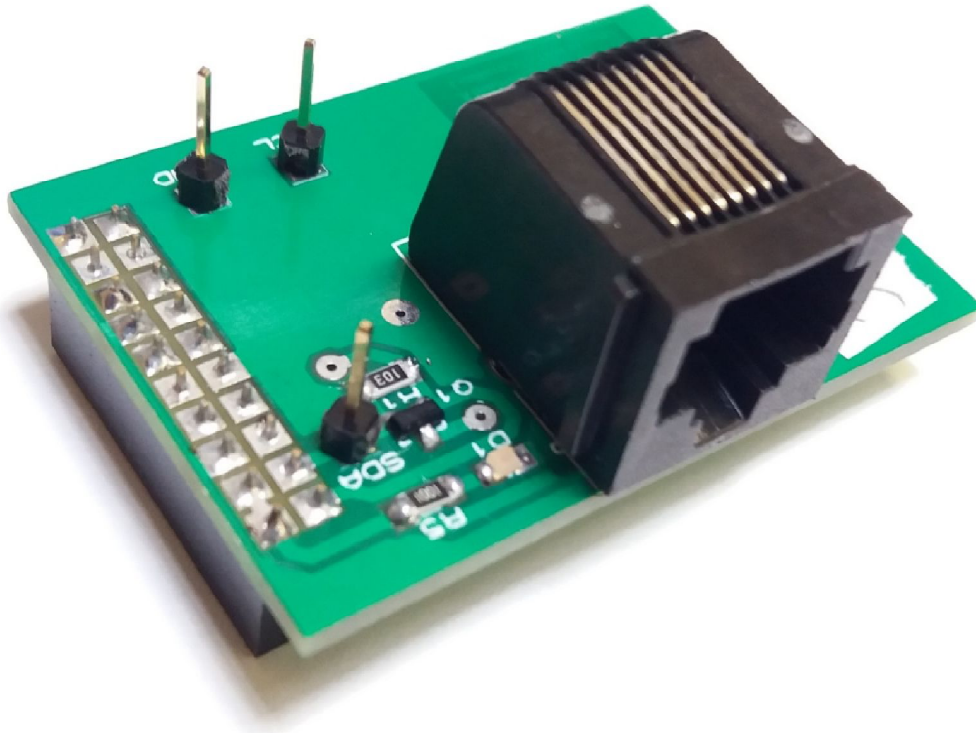
- J'ai ensuite soudé les résistances et les diodes.
- Suivi des relais.
- Ensuite les bornes qui serviront à intégrer la mini Arduino.
- Viens ensuite le circuit à 8 pattes pour transformer le 24V en 5V.
- Les pico test
- Puis la Led témoin d'alimentation et les condensateurs.

Carte avec les composants soudés. (Dans son boîtier) :



Connecteur DIN pour la commande murale.

4.5 Level Shifter (Tuto rédigé pour les membres IR)



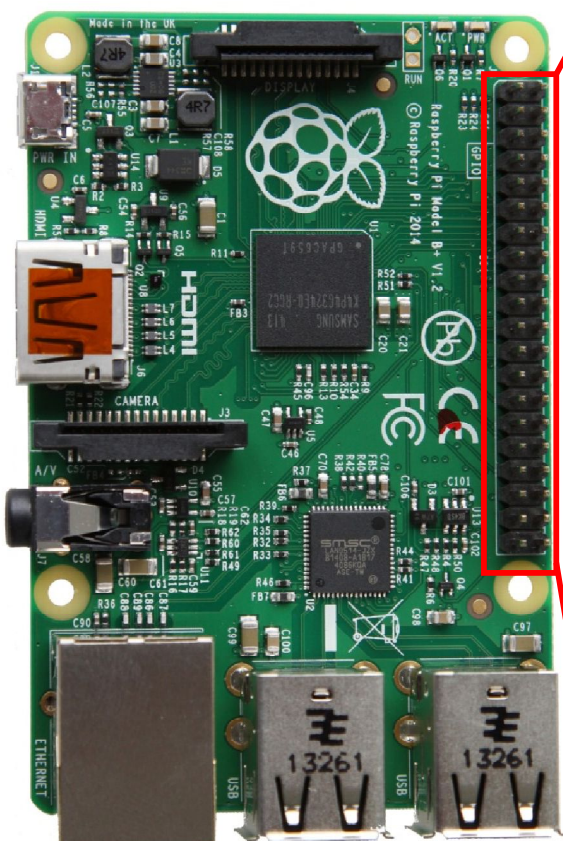
Shield Rpi level shifter RJ45

Présentation

Ce Shield est avant tout un level shifter, c'est-à-dire qu'il transforme le 3.3 V de la Raspberry en 5 V pour être compatible avec une Arduino par exemple. Il permet aussi la communication I2C entre la Raspberry et un module I2C (capteur par exemple).

Branchement

GPIO Raspberry Pi 2



GPIO#	NAME		NAME	GPIO#
	3.3 VDC Power	1		2
			5.0 VDC Power	
8	GPIO 8 SDA1 (I2C)	3		4
9	GPIO 9 SCL1 (I2C)	5		6
			Ground	
7	GPIO 7 GPCLK0	7		8
			GPIO 15 TxD (UART)	15
	Ground	9		10
			GPIO 16 RxD (UART)	16
0	GPIO 0	11		12
			GPIO 1 PCM_CLK/PWM0	1
2	GPIO 2	13		14
			Ground	
3	GPIO 3	15		16
			GPIO 4	4
	3.3 VDC Power	17		18
			GPIO 5	5
12	GPIO 12 MOSI (SPI)	19		20
			Ground	
13	GPIO 13 MISO (SPI)	21		22
			GPIO 6	6
14	GPIO 14 SCLK (SPI)	23		24
			GPIO 10 CE0 (SPI)	10
	Ground	25		26
			GPIO 11 CE1 (SPI)	11
	SDA0 (I2C ID EEPROM)	27		28
			SCL0 (I2C ID EEPROM)	
21	GPIO 21 GPCLK1	29		30
			Ground	
22	GPIO 22 GPCLK2	31		32
			GPIO 26 PWM0	26
23	GPIO 23 PWM1	33		34
			Ground	
24	GPIO 24 PCM_FS/PWM1	35		36
			GPIO 27	27
25	GPIO 25	37		38
			GPIO 28 PCM_DIN	28
	Ground	39		40
			GPIO 29 PCM_DOUT	29

Attention! The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>

Branchement du Shield

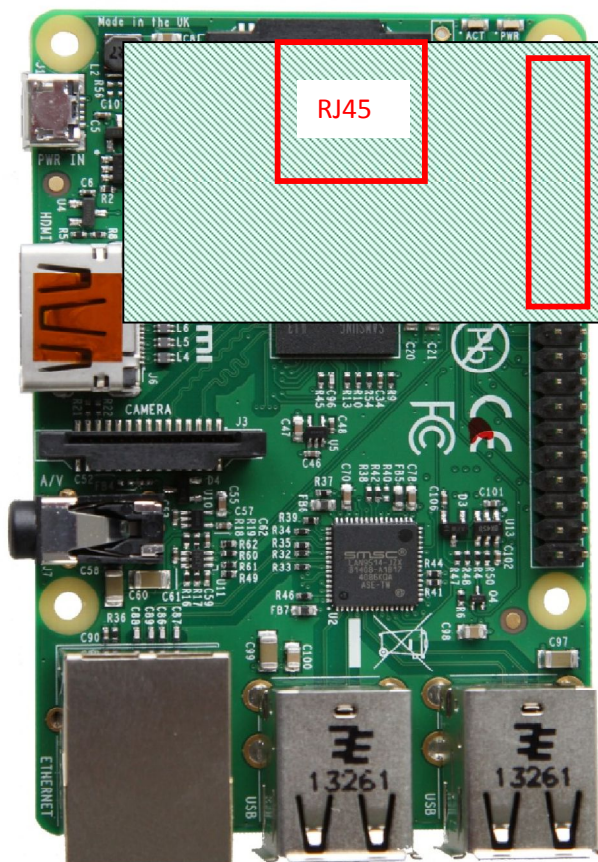


Vue de dessus



RJ45

Vue de dessous



Brancher le shield à partir de la broche 3.3V de La Rpi comme indiqué ci-contre.

Le RJ45 servira à lier le module i2c et la Raspberry Pi entre eux pour ainsi communiquer en i2c.

Installation i2c

Mettre en commentaire la ligne I2C Blacklist :

```
$ cat /etc/modprobe.d/raspi-blacklist.conf
# blacklist spi and i2c by default (many users don't need them)
blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

Installer i2c.dev dans le fichier Module :

Ajoutez le à la fin de /etc/modules

```
i2c-dev
```

Installer l'outil I2C:

```
$ sudo apt-get install i2c-tools
```

Redémarrer la Raspberry Pi. Après ça, Vous pourrez utiliser le bus i2c:

```
pi@raspberrypi ~ $ ll /dev/i2c*
crw-rw---T 1 root i2c 89, 0 May 25 11:56 /dev/i2c-0
crw-rw---T 1 root i2c 89, 1 May 25 11:56 /dev/i2c-1
```

Maintenant utilisons un test simple pour analyser le bus i2c:

Commande : `i2cdetect -y 1`

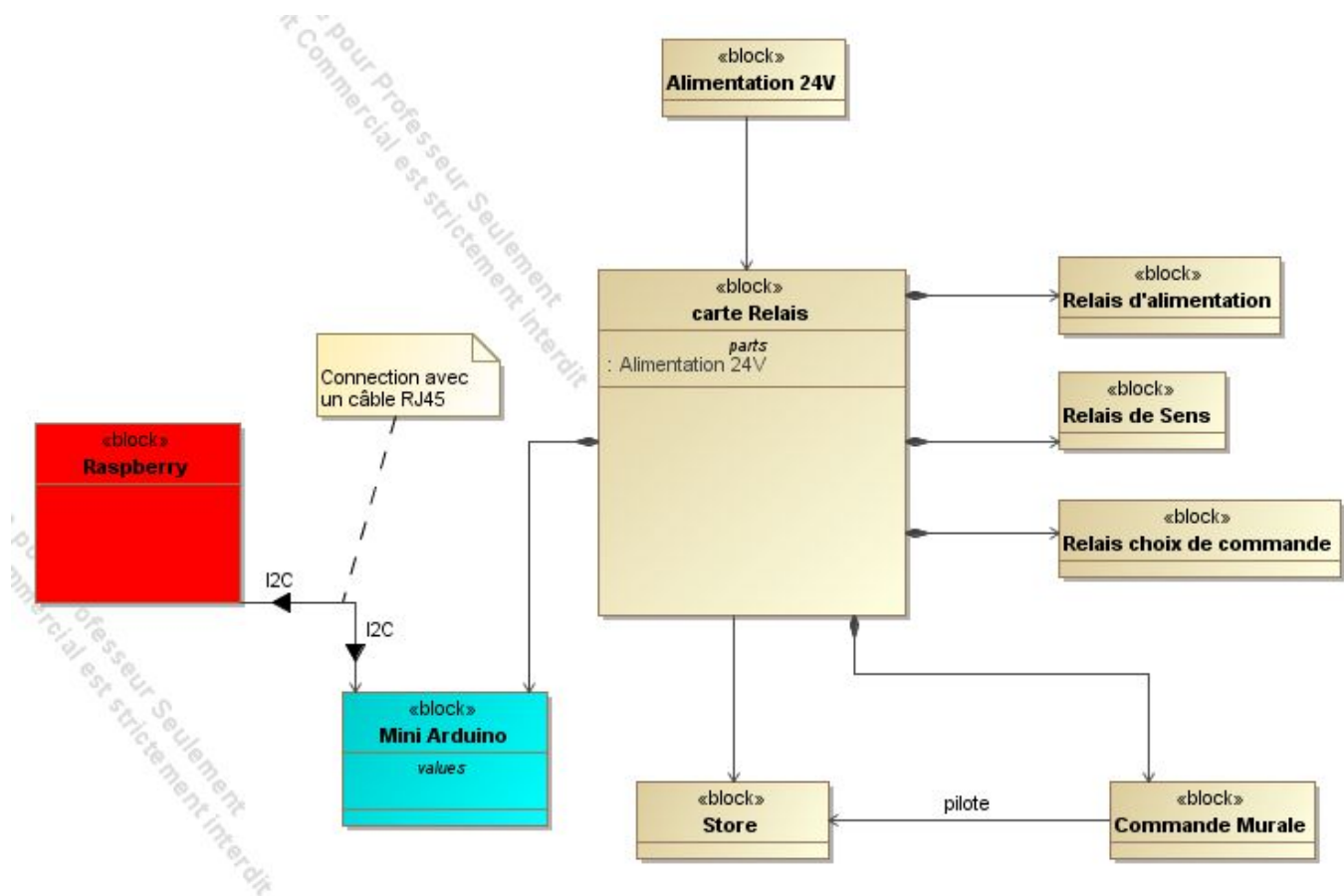
ou

`i2cdetect -y 0`

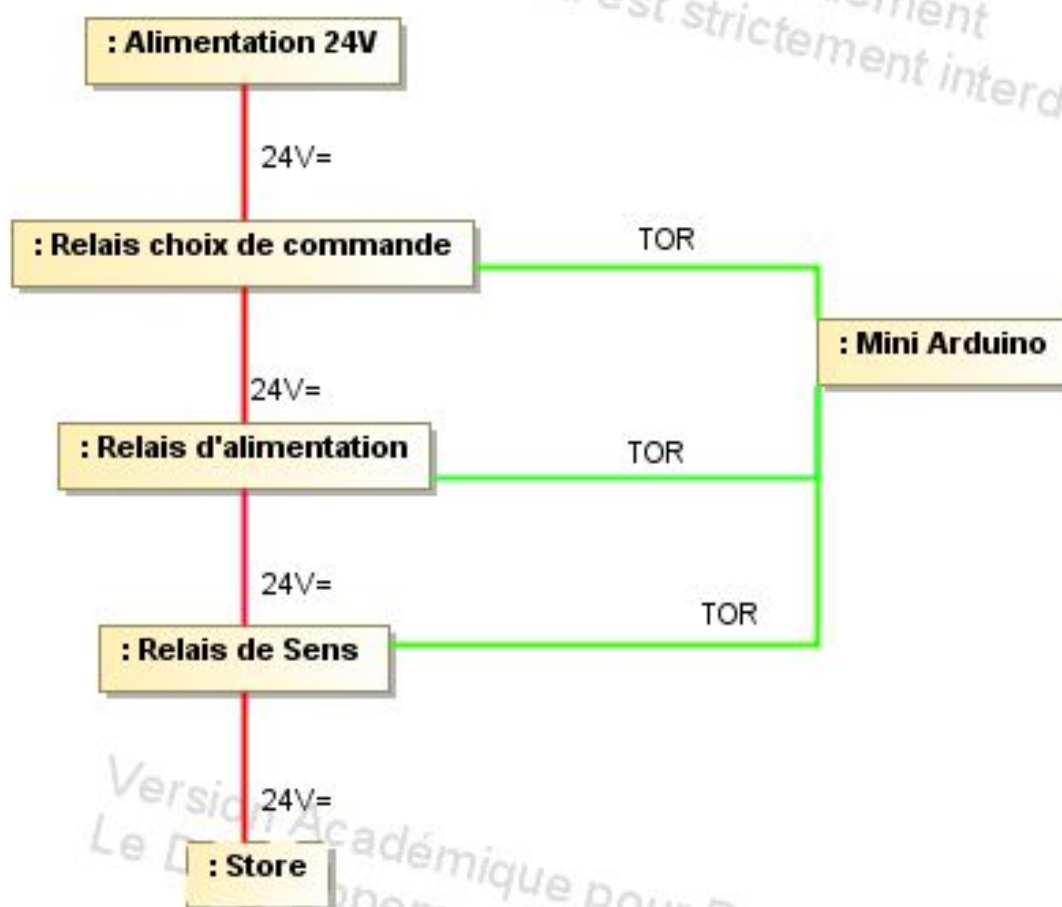
```
pi@raspberrypi ~ $ i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

4.6 Diagramme SYSML (BDD, IBD et Activités)

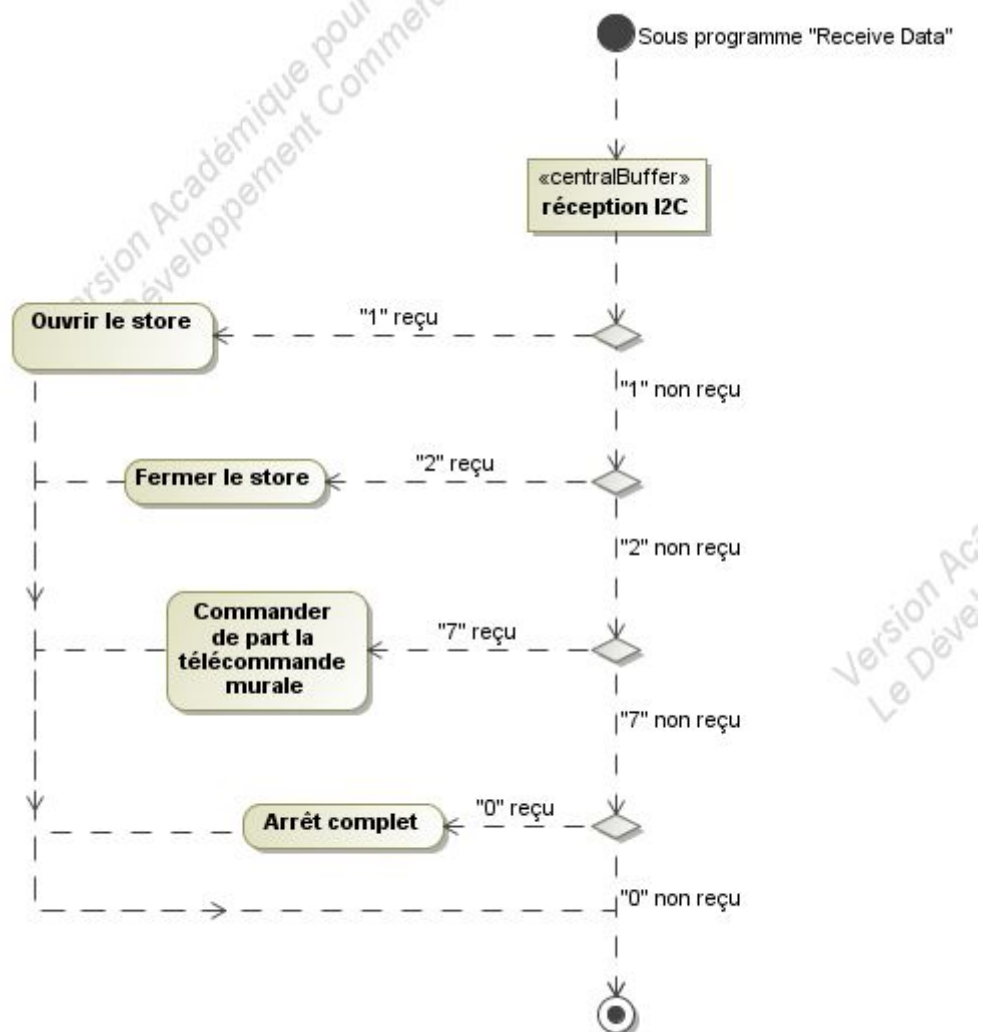
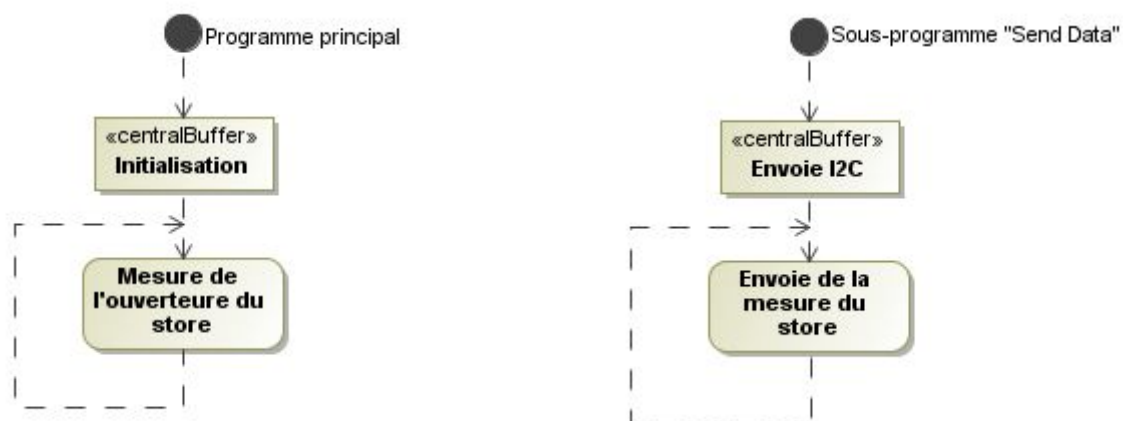
4.6.1 Diagramme de Block (BDD)



4.6.2 Diagramme de Block Interne (IBD)



4.6.3 Diagramme d'Activités



4.7 Programme final de l'Arduino

```

#include <Wire.h>
#include <Maxbotix.h>

const int on_off= 10;
const int commande_sens= 11;
const int choix_commande= 13;
char Recup;
int number;
long anVolt, inches, cm;
int sum=0;//Create sum variable so it can be averaged
int avgrange=60;//Quantity of values to average (sample size)
Maxbotix rangeSensorAD(A1, Maxbotix::AN, Maxbotix::LV, Maxbotix::BEST, 9);

int dataReceived = 0;

void setup() {
  pinMode(on_off, OUTPUT);
  pinMode(commande_sens, OUTPUT);
  pinMode(choix_commande, OUTPUT);
  Serial.begin(9600);
  Wire.begin(0x04); // Joindre le Bus I2C avec adresse #4
  Wire.onReceive(receiveData);
  Wire.onRequest(sendData);
  Serial.println("Ready!");
}

void loop() {
  delay(100);
}

void receiveData(int byteCount){ // données reçu de part le bus I2C
  while(Wire.available()) {
    dataReceived = Wire.read();

    Recup = dataReceived ;

    if (Recup == '1') // si "1" est reçu alors le store descend
    {
      digitalWrite(on_off, HIGH);
      digitalWrite(commande_sens, LOW);
      digitalWrite(choix_commande, LOW);
    }
    else if (Recup == '0') // si "0" est reçu alors tout est éteint
    {
      digitalWrite(on_off, LOW);
      digitalWrite(commande_sens, LOW);
      digitalWrite(choix_commande, LOW);
    }
    else if (Recup == '2') // si "2" est reçu alors le store monte
    {
      digitalWrite(on_off, HIGH);
      digitalWrite(commande_sens, HIGH);
      digitalWrite(choix_commande, LOW);
    }
  }
}

```

```

else if (Recup == '7') // si "7" est reçu commande murale prend le relais
{
    digitalWrite(choix_commande, HIGH);
    digitalWrite(on_off, LOW);
    digitalWrite(commande_sens, LOW);
}

    Serial.print("I received: ");
    Serial.println(Recup);
}
    unsigned long start;

if (rangeSensorAD.getRange() < 183) // mesure de la distance du store
{
    start = millis();
    Serial.print("AD (BEST): ");
    Serial.print(rangeSensorAD.getRange());
    Serial.println(" cm");
    delay(500);
}
    else{}
        number = rangeSensorAD.getRange();

        }

void sendData(){ // mesure envoyé
    Wire.write(number);
}

```

4.8 Programme Qt (mainwindow.cpp)

```

MainWindow MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)

{
    int erreur;
    ui->setupUi(this);
    wiringPiSetupGpio();
    fd=wiringPiI2CSetup(0x04); // envoie de données à l'adresse 0x04
    erreur = wiringPiI2CRead(fd);
    if(erreur < 0 ) // test de connexion dès le lancement du programme

    {
        ui->label->setText("erreur connexion");
        ui->label->setStyleSheet("color: #FF0000");
    }

    else

    {
        ui->label->setText("connecte");
        ui->label->setStyleSheet("color: #096A09");
    }

    ui->lcdNumber->setNumDigits(4);
}

MainWindow::~~MainWindow()

{
    delete ui;
}

void MainWindow::on_pushButton_clicked() // test de connexion à l'aide
d'un bouton

{
    int erreur;
    erreur = wiringPiI2CRead(fd);

    if(erreur < 0 )

    {
        ui->label->setText("erreur connexion");
        ui->label->setStyleSheet("color: #FF0000");
    }

    else

    {
        ui->label->setText("connecte");
        ui->label->setStyleSheet("color: #096A09");
    }
}

```

```

}

void MainWindow::on_radioButton_clicked()
{
    wiringPiI2CWrite(fd, 0x31);      // envoie d'un "1"
}

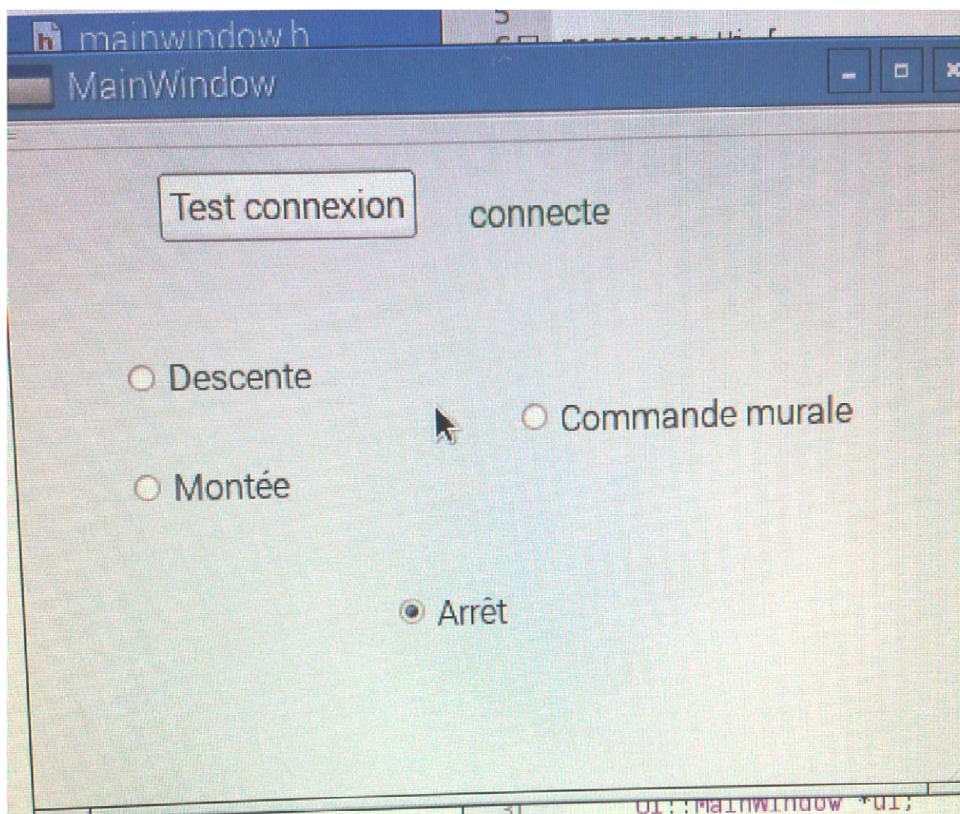
void MainWindow::on_radioButton_2_clicked()
{
    wiringPiI2CWrite(fd, 0x32);      // envoie d'un "2"
}

void MainWindow::on_radioButton_3_clicked()
{
    wiringPiI2CWrite(fd, 0x30);      // envoie d'un "0"
}

void MainWindow::on_radioButton_4_clicked()
{
    wiringPiI2CWrite(fd, 0x37);      // envoie d'un "7"
}

```

4.8.1 Apperçu de l'application Qt



4.9 Annexes

4.9.1 Journal de travail :

11/01/2016: répartition des tâches

13/01/2016: exemple de programme Arduino pour l'I2c

15/01/2016: test du store (1 mètre déplié) fil noir/blanc controle moteur. noir ouvre le store / blanc ferme le store
Test du relais

18/01/2016: Montage schéma Proteus. Paramétrage de base Rpi.

20 et 22/01/2016: mise en service du montage (sans Arduino) avec 2 relais.

25/01/2016: recherche d'un programme Arduino

27/01/2016: Test des transistor + Arduino

29/01/2016: test programme Arduino (concluant)

03/02/2016: activation i2c raspberry. test communication dans un sens (raspberry → Arduino)

05/02/2016: test capteur ultrason. (LV-MAXsonar-EZ);

22/02/2016: changement de store ok. programme Arduino + capteur ok

24/02/2016-11/03/2016: préparation revue de projet n°1 + soudure level shifter

14/03/2016: soudure level shifter + tuto du shield pour IR

16/03/2016: Schema isis et routage de la carte

20/04/2016: Fabrication de la carte.

22 - 25/04/2016: Soudure

27 - 29/04/2016: rédaction du projet

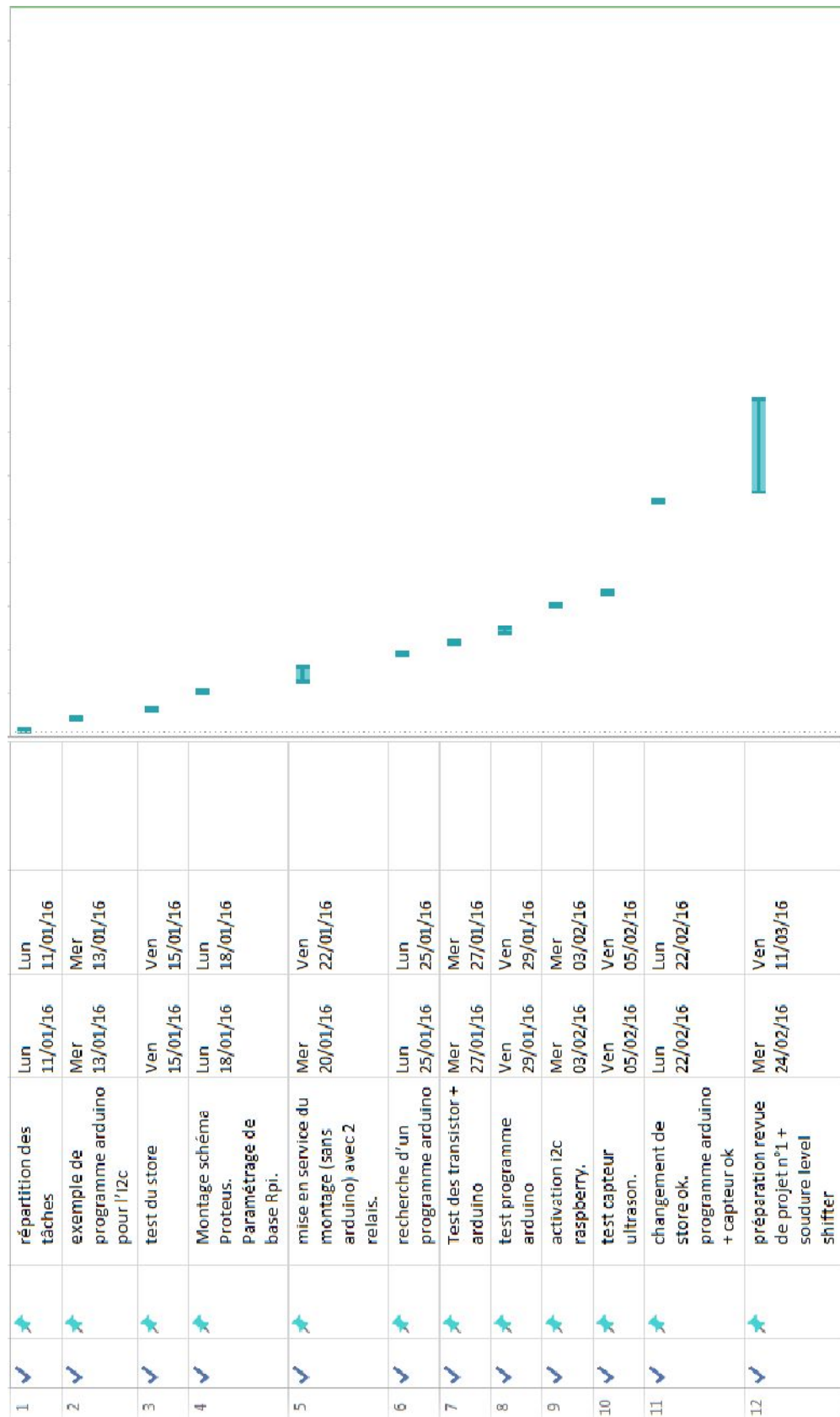
2/05/2016: test et réparation de la carte

4/05/2016: Branchement du capteur et de la prise pour commande murale

9/05/2016: Programme capteur ultrason. Arduino

11 - 20/05/2016: programme Qt commande store et commande murale

4.9.2 Gantt



13	✓	★	soudure level shifter + tuto du shield pour IR	Lun 14/03/16	Lun 14/03/16	
14	✓	★	Schema isis et routage de la carte	Mer 16/03/16	Mer 16/03/16	
15	✓	★	Fabrication de la carte.	Mer 20/04/16	Mer 20/04/16	
16	✓	★	Soudure des composant	Ven 22/04/16	Lun 25/04/16	
17	✓	★	rédaction du projet deuxième revue	Sam 27/02/16	Lun 29/02/16	
18	✓	★	test et réparation de la carte	Lun 02/05/16	Lun 02/05/16	
19	✓	★	Branchement du capteur et de la prise pour commande murale	Mer 04/05/16	Mer 04/05/16	
20	✓	★	Programme capteur ultrason . arduino	Lun 09/05/16	Lun 09/05/16	
21	✓	★	programme Qt commande store et commande murale	Mer 11/05/16	Ven 20/05/16	
22		★?	Programme Qt capteur ultrason	Lun 23/05/16		

4.10 Conclusion

A l'heure actuelle, ma partie dans ce projet tel qu'elle est présenté dans ce dossier, fonctionne bien. Tout mes programmes, en ce qui concerne la commande du store qu'elle soit de part la commande murale ou depuis la Raspberry, fonctionne. Il n'y a qu'une seule chose que je n'ai pas mise en marche avant de rendre ce dossier, c'est la captation de la distance du store. En espérant pouvoir finir cette tâche avant le 16 Juin.

5 Partie personnelle (IR1)

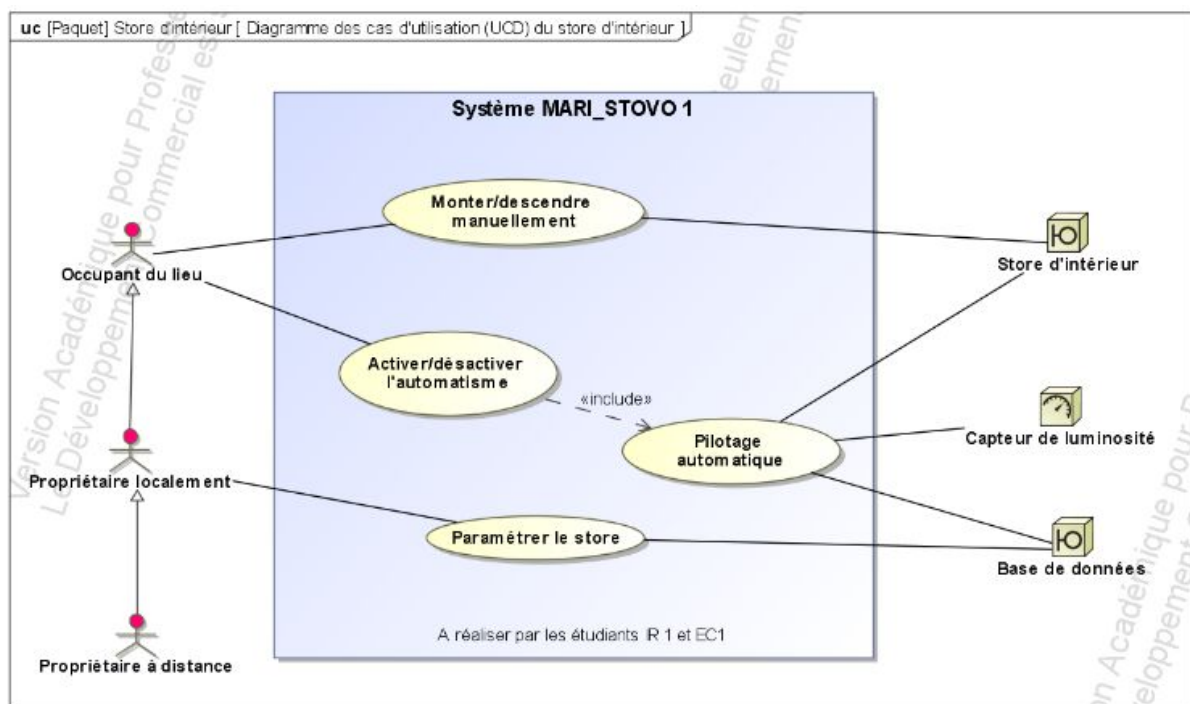
CHASSILLAN Bastien

5.1 Présentation du store d'intérieur :

	Fonctions à développer et tâches à effectuer	
Étudiant 1	Liste des fonctions assurées par l'étudiant	Installation : Qt, MySql, i2c-dev.
IR 1	Gestion du store d'intérieur avec son détecteur de luminosité.	Mise en œuvre : i2c-dev. Pilotage du store via le bus I2C et le PCF8574. Acquisition de la luminosité du TSL2561. Configuration : RPI Réalisation : Pilotage du store selon les consignes de la BDD et de la luminosité. Documentation : Installation, mise en service, dossier de développement.

Ma partie de projet consiste à contrôler un store d'intérieur en fonction de la lumière.

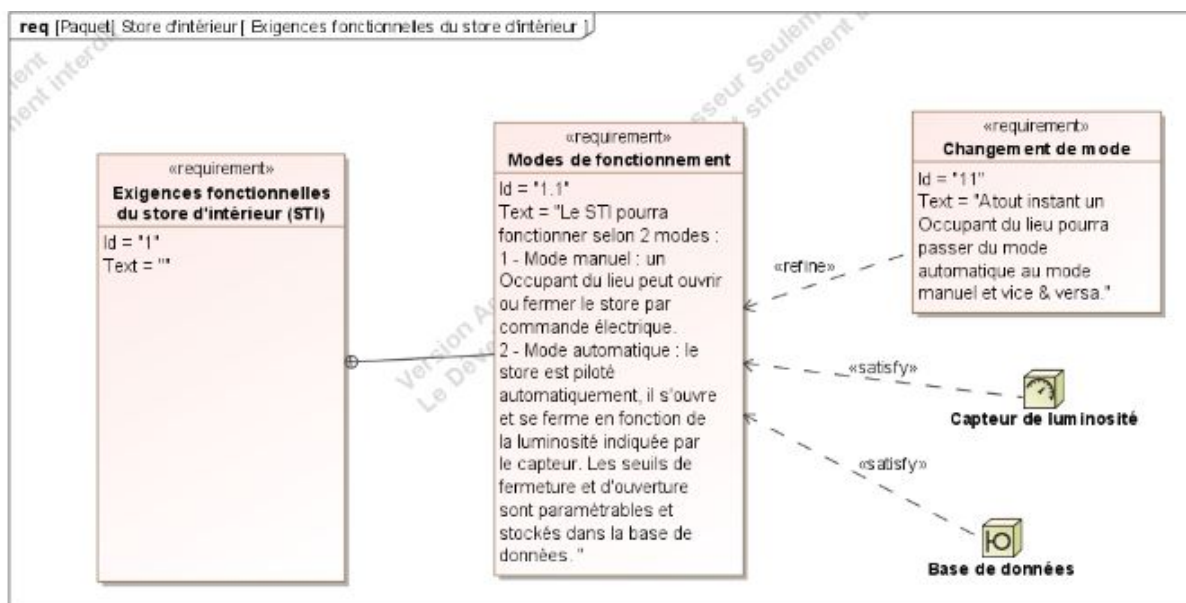
Cette tâche peut-être représentée par le diagramme de cas d'utilisation ci dessous :



De plus l'utilisateur aura la possibilité basculer entre un mode "Automatique", qui contrôlera la position du store en fonction des consignes de lumière et le mode "Manuel" qui permettra de commander l'appareil manuellement selon sa convenance.

L'occupant peut modifier des seuils qui seront exprimé en Lux. Ses seuils seront envoyés dans la base de données et seront stockés.

Le store d'intérieur , le capteur de lumière et le programme communiquerons à l'aide du protocole I2C, grâce à la bibliothèque « i2c-dev » que j'ai développé.



5.2 Présentation du matériel :

5.2.1 Raspberry + Level Shifter



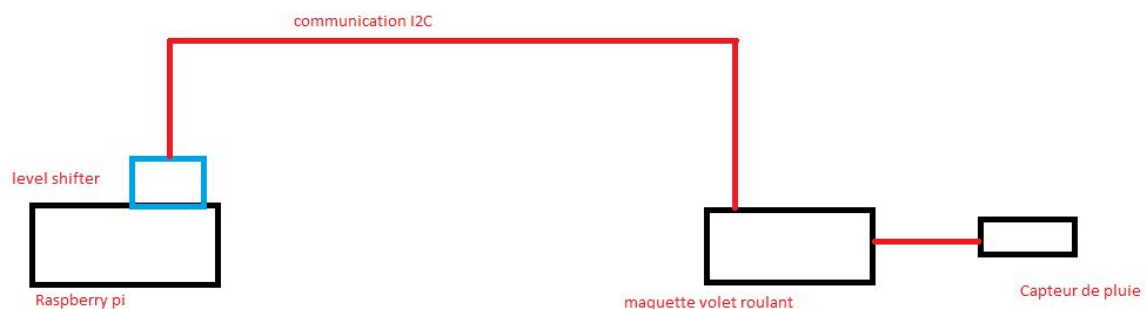
Le projet sera sur une raspberry, sur laquelle sera installé un level shifter, convertissant la tension de sortie des broches utilisant le Bus I2C de 3,3 Volts à 5 Volts.

La présence du level shifter est dû à la tension de fonctionnement du capteur de lumière qui est de 5 Volts

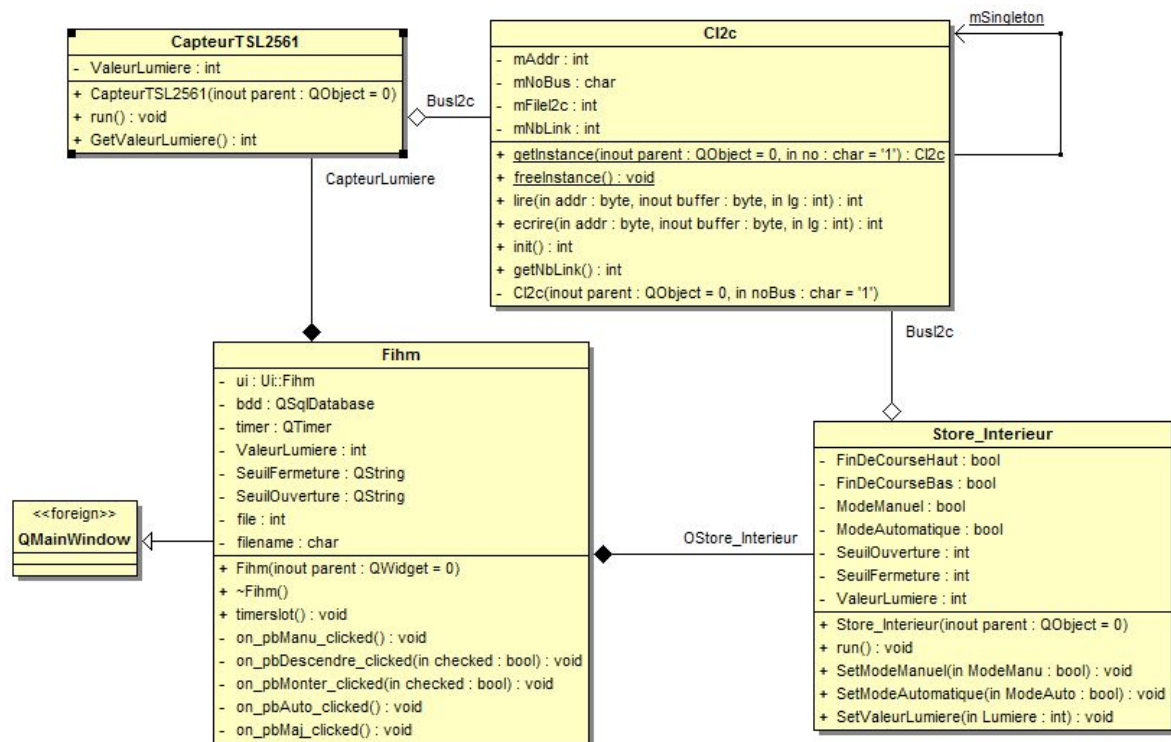
5.2.2 Maquette Vollet

A la sortie du level shifter, le premier appareil présent sur le Bus sera le Store d'interieur

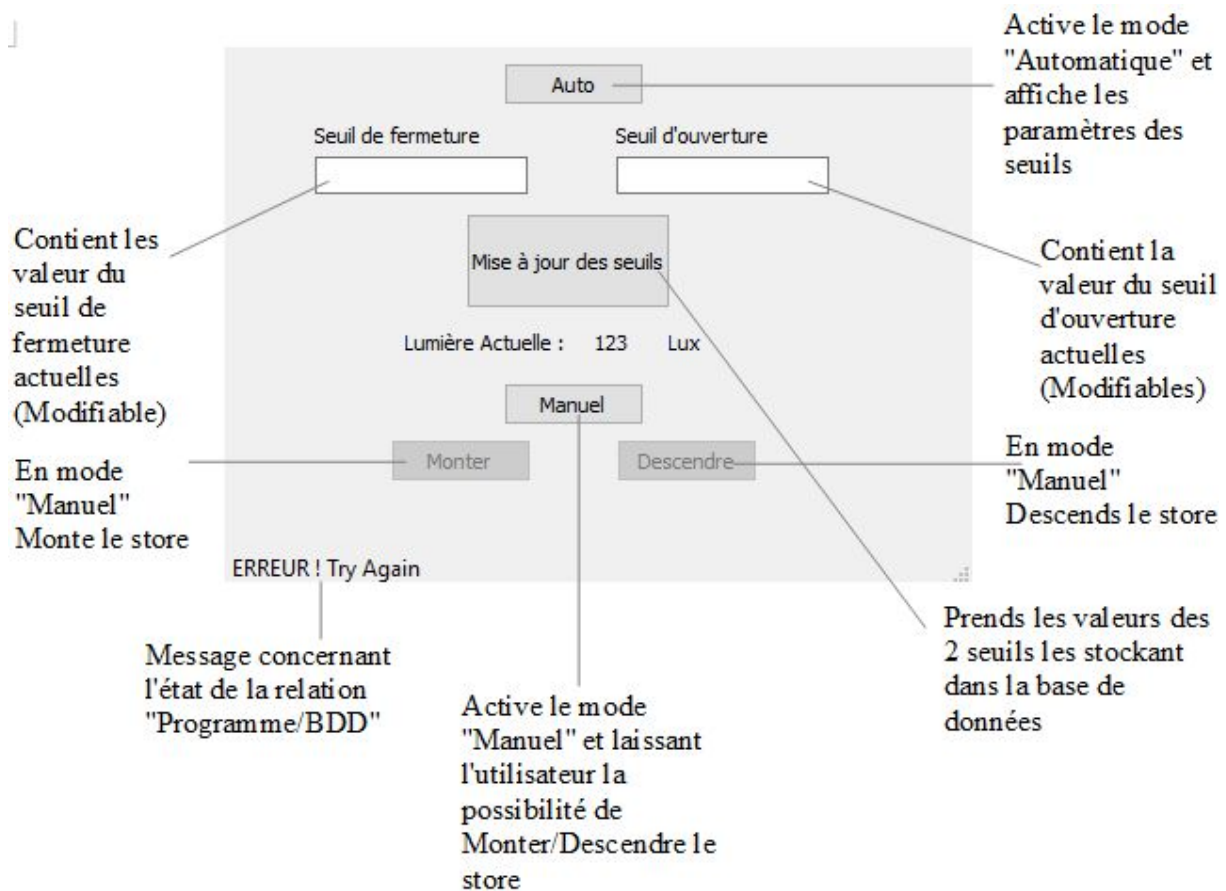
Le level shifter nous sert aussi d'interface du Bus I2C et aura la forme suivante :



5.3 Diagramme UML et Programme :



Pour que l'utilisateur puisse manipuler le store, une interface graphique a été implémentée :



Les lignes des seuils sont modifiables, et seront appliquer au programme quand le bouton de mise à jour sera appuyé.

Lors du lancement de l'interface, les lignes afficherons les paramètres actuels du mode "Automatique".

5.4 Extraits du travail réaliser sur le programme :

Fihm.cpp

Connexion à la base de données avec test

```
Fihm::Fihm(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Fihm)
{
    ui->setupUi(this);
    db = QSqlDatabase::addDatabase("QMYSQL");
    db.setDatabaseName("MARIE-STOVO");
    db.setUserName("root");
    db.setPassword("admin");
    if ( ! db.open() )
    {
        QString mess;
        ui->statusBar->showMessage("ERREUR ! Try Again");
        return;
    }
}
```

Fihm.h

Liste des attributs et méthodes ainsi que des bibliothèques

```
23 private:
24     Ui::Fihm *ui;
25     QSqlDatabase bdd;
26     QTimer timer;
27     QSqlQuery query;
28     QString lightvalue;
29
30 public slots:
31     void timerslot();|
32
33 private slots:
34     void on_pbMonter_clicked(bool checked);
35
36     void on_pbDescendre_clicked(bool checked);
37
38     void on_pbManu_clicked();
39
40     void on_pbAuto_clicked();
41 };
42
43 #endif // FIHM_H
44
```

Fihm.cpp

Connexion à la base de données avec test

```

Fihm::Fihm(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Fihm)
{
    ui->setupUi(this);
    db = QSqlDatabase::addDatabase("QMYSQL");
    db.setDatabaseName("MARIE-STOVO");
    db.setUserName("root");
    db.setPassword("admin");
    if ( ! db.open() )
    {
        QString mess;
        ui->statusBar->showMessage("ERREUR ! Try Again");
        return;
    }
}

```

Fihm.h

Liste des attributs et méthodes ainsi que des bibliothèques

```

1  #ifndef FIHM_H
2  #define FIHM_H
3
4  #include <QMainWindow>
5  #include <QtSql/qsqldb.h>
6  #include <QtSql/QSqlDatabase>
7  #include <QtSql/QSqlDriver>
8  #include <QtSql/QSqlQuery>
9  #include <QTimer>
10
11 namespace Ui {
12 class Fihm;
13 }
14
15 class Fihm : public QMainWindow
16 {
17     Q_OBJECT
18
19 public:
20     explicit Fihm(QWidget *parent = 0);
21     ~Fihm();
22
23 private:
24     Ui::Fihm *ui;
25     QSqlDatabase bdd;
26     QTimer timer;
27     QSqlQuery query;
28     QString lightvalue;
29
30 public slots:
31     void timerslot();
32
33 private slots:
34     void on_pbMonter_clicked(bool checked);
35
36     void on_pbDescendre_clicked(bool checked);
37
38     void on_pbManu_clicked();
39
40     void on_pbAuto_clicked();
41 };
42
43 #endif // FIHM_H
44

```

Le timer sert à afficher chaque seconde la valeur lumineuse de la pièce.
 Algorithme du store d'intérieur :

```

9  void Store_Interieur::run()
10 {
11     while(1)
12     {
13         while(ModeAutomatique == true)
14         {
15             if(SeuilOuverture > ValeurLumiere)
16             {
17                 while(FinDeCourseHaut)
18                 {
19                     //Monter le store
20                 }//while
21             }//if
22
23             if(SeuilFermeture < ValeurLumiere)
24             {
25                 while(FinDeCourseBas == false)
26                 {
27                     //Descendre le store
28                 }//while
29             }//if
30             qDebug("Je suis auto");
31             sleep(1);
32         }//while
33
34         while(ModeManuel== true)
35         {
36             qDebug("Je suis manu");
37             sleep(1);
38         }//while
39     }//while
40
41 }//run
42
43

```

Lors du développement, j'ai rencontré beaucoup de difficulté à manipuler le Bus I2C. J'ai pris du retard sur l'avancement de mon projet.

Je n'ai pas encore placé les mutex afin d'empêcher un accès simultané au Bus. Le but est de protéger les trames qui circulent, et éviter des erreurs.

Le capteur et le store sont indépendants et ne possèdent aucun lien entre eux. Afin d'accéder aux données, j'utilise des accesseurs.

Exemples :

```
51 void Store_Interieur::SetModeManuel (bool ModeManu)
52 {
53     ModeManuel = ModeManu;
54 }
55 //SetModeManuel
```

Setteur = La variable voulue sera donnée en paramètre de la fonction qui stockera dans une variable privée.

```
24 int CapteurTSL2561::GetValeurLumiere ()
25 {
26     return ValeurLumiere;
27 }
```

Getteur = On retourne en résultat le contenu voulu que l'on met dans une variable privée dans la classe désirée.

La Classe de gestion de l'I2C est créée en suivant le principe du patron de construction « Singleton ». Le singleton, sert à empêcher la création de plusieurs objets identiques, leur donnant à la place l'adresse de l'objet déjà créé.

Pour empêcher cette création, on met le constructeur en « Private », donc pour activer celui-ci, on appelle la fonction suivante :

```

CI2c *CI2c::getInstance(QObject *parent, char no)
{
    if (mSingleton == NULL)
    {
        qDebug("L'objet CI2c sera créé !");
        mSingleton = new CI2c(parent, no);
        mSingleton->init();
        mSingleton->mNbLink++;
    }
    else
    {
        qDebug("singleton already created!");
    }
    return mSingleton;
} // getInstance

```

"mSingleton" créé grâce à init que si l'objet n'as pas déjà été crée, et donne à la place l'adresse de l'objet.

L'init du bus est en faite une ouverture de fichier :

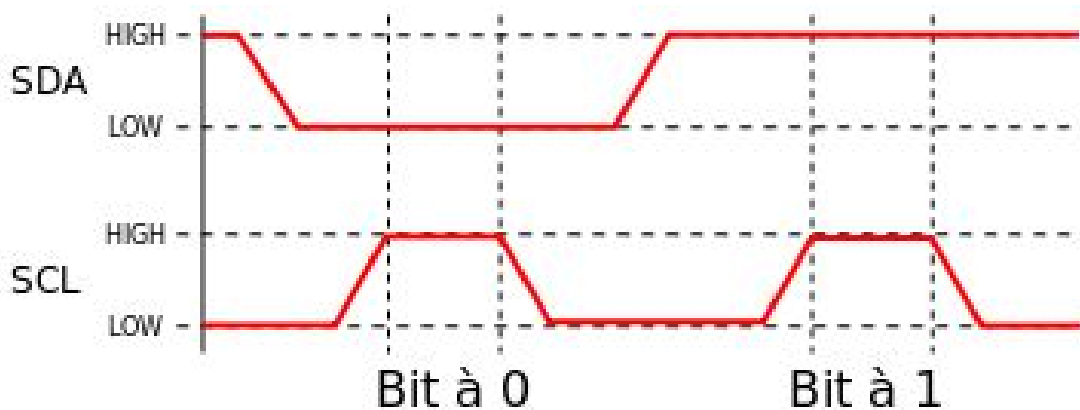
```

int CI2c::init()
{
    char filename[20];
    qDebug() << mNoBus;
    sprintf(filename, "/dev/i2c-%c", mNoBus);
    if((mFileI2c=open(filename, O_RDWR))==-1) { // ouvre
        qDebug("Erreur ouverture acces au bus I2C");
        return -1;
    } // if open
    return mFileI2c;
} // init

```

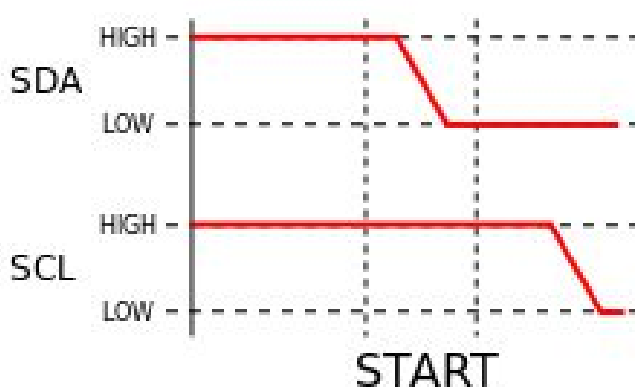
I2C (signifie : Inter-Integrated Circuit, en anglais) est un Bus informatique qui a émergé de la « guerre des standards » lancée par les acteurs du monde électronique. Conçu par Philips pour les applications de domotique et d'électronique domestique, il permet de relier facilement un microprocesseur et différents circuits, notamment ceux d'une télévision moderne : récepteur de la télécommande, réglages des amplificateurs basses fréquences, tuner, horloge, gestion de la prise péritel, etc

Bus i2c utilise un codage NRZ.

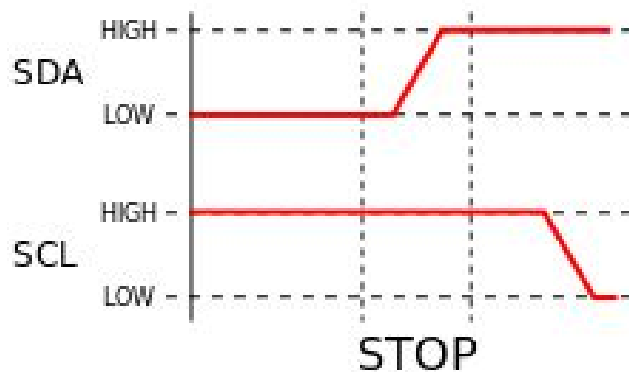


Le SDA doit être stable lors des passage à 1 de SCL

BIT DE START



BIT DE STOP



5.5 CONCLUSION

5.5.1 Difficultés rencontrées

Durant le projet, du retard a été engrangé suite à la compréhension et le développement de la classe s'occupant de manipuler le Bus I2C.

Une fois ce développement mis au point, j'ai eu énormément de mal à comprendre les octets de données que je devais envoyer pour contrôler les appareils du système.

Suite à ce retard, j'ai été dans l'incapacité de rendre portable la classe d'I2C.

De plus, j'ai passé du temps à faire les demandes à la bases de données, dû à une erreur inconnue avec l'étudiant en charge de la base de données, que j'ai dû finalement gérer moi même.

Le dernier problème croisé est suite à l'intégration, car non ne possèdent pas 3 volets comme défini dans le cahier des charge, et donc nous allons porter des modifications pour que le projet final puisse fonctionner, ajoutant du travail supplémentaire.

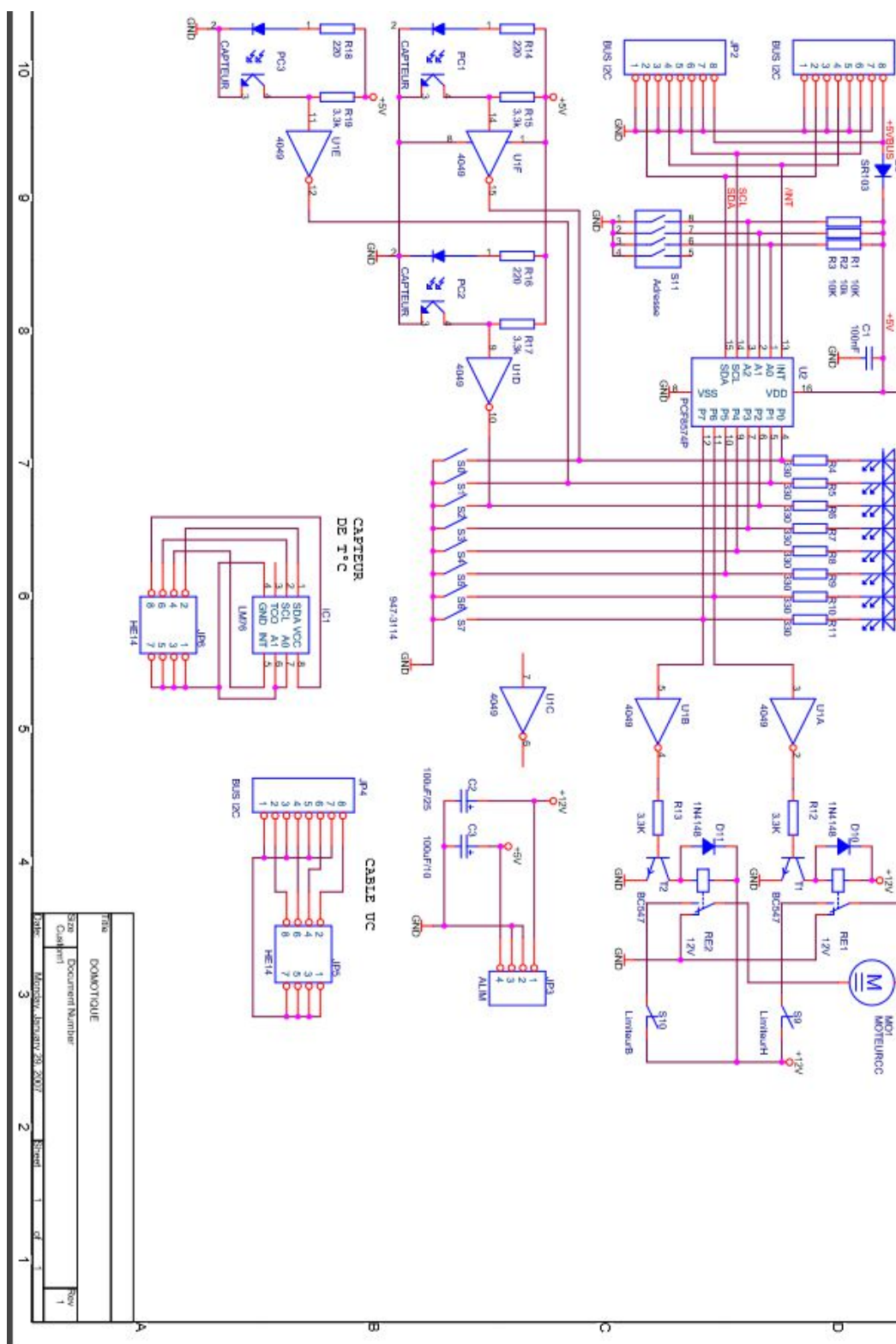
5.5.2 Pour conclure

J'ai éprouvé beaucoup de difficultés pour avancer dans se projet, mais aussi j'ai énormément appris sur la programmation et de l'ambiance de travail au sein d'un projet.

J'ai aussi pris énormément de plaisir à écrire et travailler sur ce Projet

5.6 Annexes

5.6.1 Schéma électronique du Store



5.6.2 Journal de bord

18 Janvier – Remise du cahier des charges + installation de logiciel pour la Raspberry

Jusqu'au 8 février – Installation de logiciels multiples pour la Raspberry

15 février jusqu'au 22 février – Réflexion et établissement d'un gant prévisionnel

24 février – Réalisations sur les requêtes SQL et définition de l'interface homme-machine

26 février – Problèmes de connectivité avec la BDD

2 mars - Problème non réglé, passage à une BDD locale, voir à l'intégration

Du 4 au 28 mars – Étude du protocole I2C

30 mars au 6 avril – Développement de la classe du BUS I2C

Le reste du mois d'avril – Développement du Thread du capteur de lumière et communication avec l'interface Homme-Machine

Mois de mai – Création et utilisation d'un algorithme gérant la volet en fonction de la lumière

Création de passerelle de communication avec des accesseurs.

6 Partie personnelle (IR2)

COLUS Julia

6.1 Présentation :

Dans le cadre de mon bts une partie du projet Mari Stovo m'a été confié. Le projet requière le fonctionnement des stores bannes et d'un volet roulant. Cette mission consiste à devoir faire fonctionner un volet roulant par rapport à la demande du cahier des charges.

Etant l'IR2 j'ai eu plusieurs taches à accomplir :

Étudiant 2	<i>Liste des fonctions assurées par l'étudiant</i>	Installation : Qt, MySql, i2c-dev.
IR 2	Gestion du volet roulant avec son détecteur de pluie.	Mise en œuvre : MySql. Pilotage du volet (PCF8574), détecteur de pluie (arduino).
		Configuration : MySql.
		Réalisation : Pilotage du volet selon les consignes de la BDD et de la pluie.
		Documentation : Installation, mise en service, dossier de développement.

J'ai donc comme tache la gestion du volet roulant avec son détecteur de pluie.

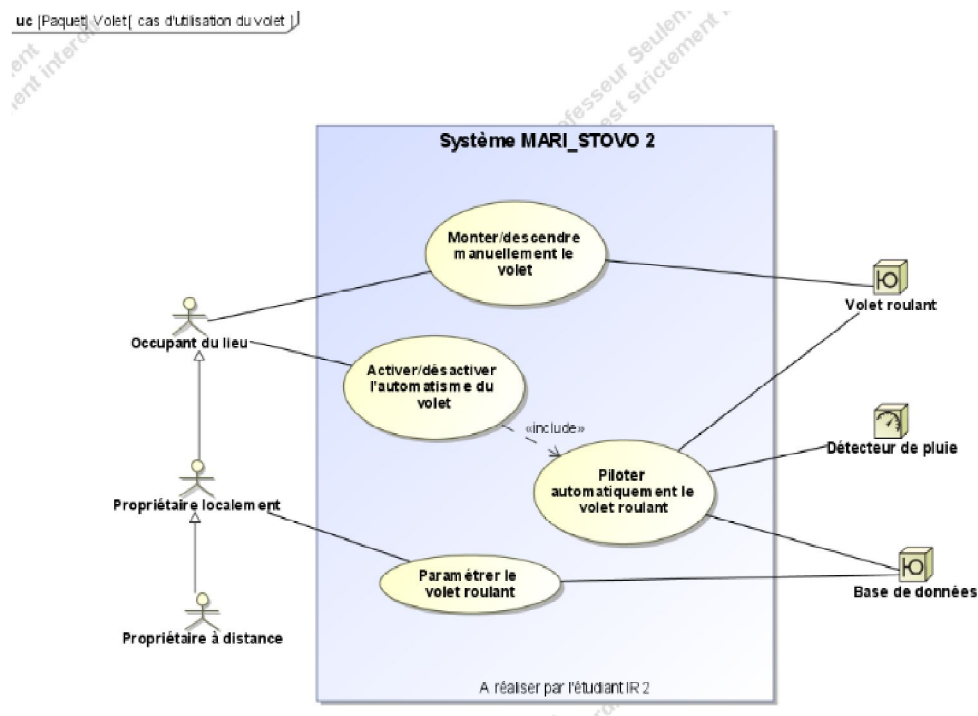
Pour travailler sur ce projet j'ai du installer les logiciel Qt créator , MySql et comprendre la communication I2C.

Pour travailler nous avons eu des cartes Raspberry Pi. Une des cartes m'a été attribuée.

La réalisation de mon projet va donc être le pilotage du volet roulant selon les consignes de la base de données et de la pluie.

Pour mieux comprendre mon projet plusieurs diagrammes ont été réalisés.

6.2 Diagramme de cas d'utilisations :



Le diagramme des cas d'utilisations va servir à donner une vision globale du comportement fonctionnel du projet.

On peut voir que l'occupant du lieu peut monter et descendre manuellement le volet roulant.

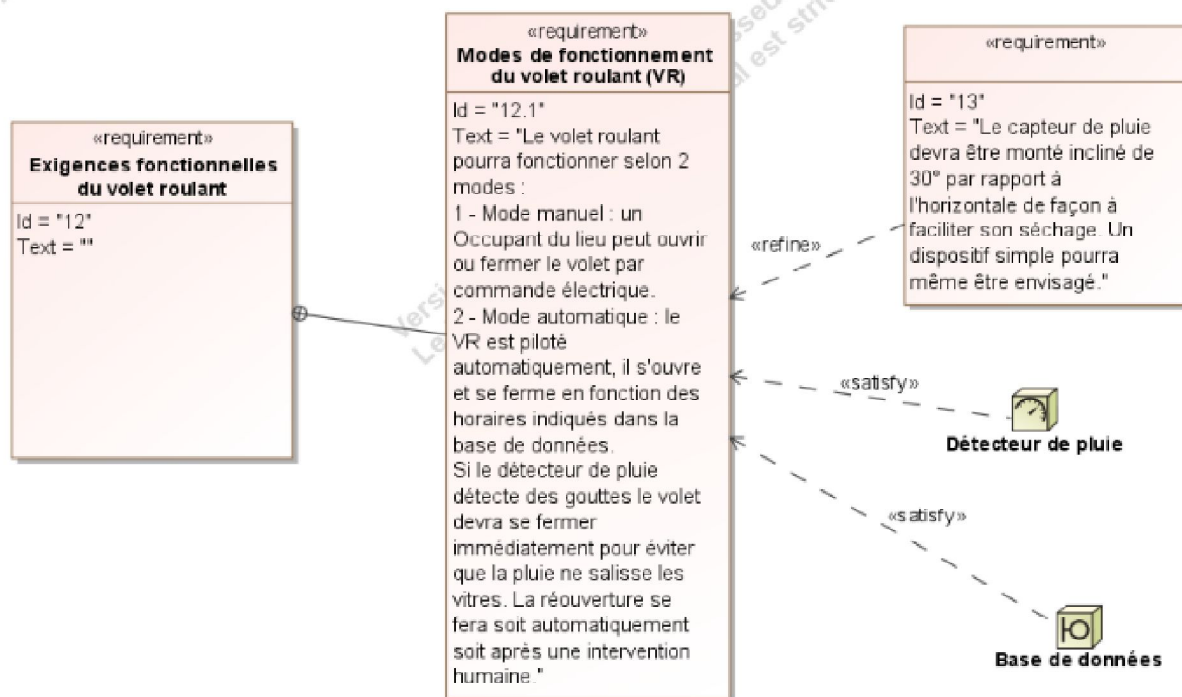
Il peut aussi activer le mode manuel et le désactiver.

Cela inclut que le pilotage du volet roulant va se faire avec la base de données qui va paramétrer les voler roulants et le détecteur de pluie qui va permettre de le monter et le descendre.

Il y aura donc deux modes :

Le mode automatique et le mode manuel.

6.3 Diagramme d'exigences :



Voici le diagramme d'exigence réalisé.

Il explique le mode de fonctionnement du volet roulant

Nous voyons ici que le volet à bien de mode.

Le mode manuel va permettre à l'occupant du lieu de pouvoir monter et descendre le volet roulant manuellement par commande électrique.

Le mode automatique va pouvoir piloter le volet roulant automatiquement.

Il va devoir s'ouvrir automatiquement à l'heure indiquée dans la base de données.

Ces horaires vont pouvoir être paramétrés par l'occupant du lieu.

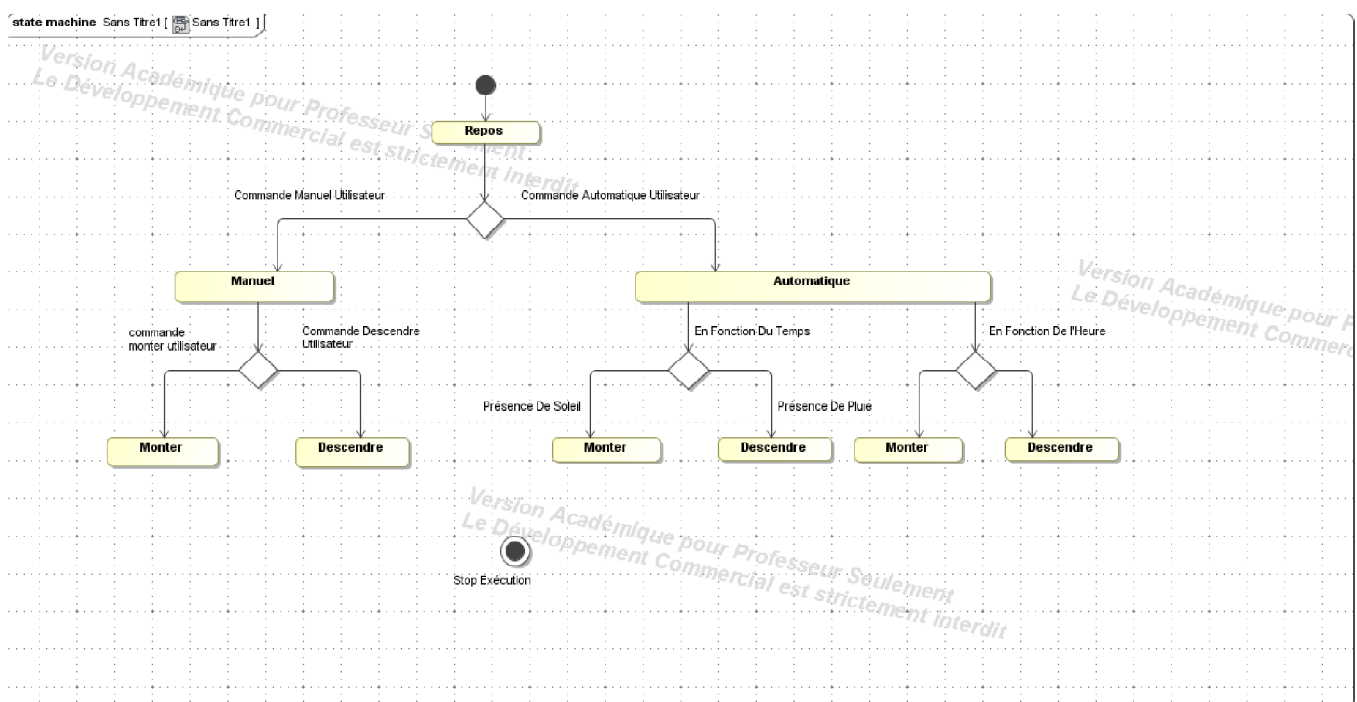
En cas de pluie le volet va devoir se fermer automatiquement pour éviter qu'il se salisse.

Il va se rouvrir quand il y aura plus de pluie ou par commande manuel

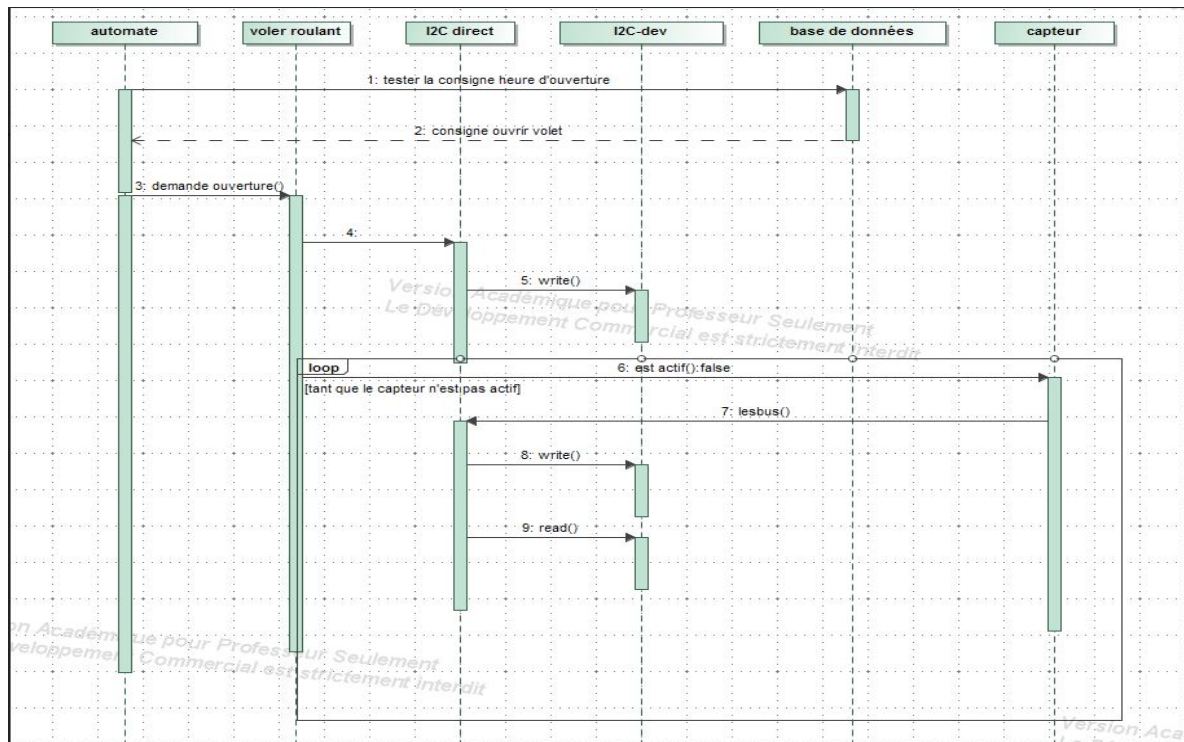
Pour ce la j'ai un détecteur de pluie.

Le détecteur de pluie a pour exigence d'être incliné à 30° par rapport à l'horizontale pour sécher plus vite.

6.4 Diagramme d'état :



6.5 Diagramme de séquences :



6.6 Programme base de données :

Pour commencer mon projet j'ai donc pris l'initiative de faire un programme sous Qt créateur pour pouvoir récupérer des données dans une base de données.

Pour cela j'ai voulu récupérer les données d'hydrométries dans la base de données du projet.

Les données d'hydrométries sont : la mesure de l'étendu d'eau.

Cela va permettre de relever le niveau de pluie.

J'ai donc commencé à créer un projet sur Qt-Créateur qui est : mari-stovo-volet

Voici le .h du programme réalisé :

```
#ifndef FIHM_H
#define FIHM_H

#include <QMainWindow>
#include <QtSql/qsqldb.h>
#include <QtSql/QtSqlDatabase>
#include <QtSql/QtSqlDriver>
#include <QtSql/QtSqlQuery>
#include <QTimer>

namespace Ui {
class fihm;
}

class fihm : public QMainWindow
{
    Q_OBJECT

public:
    explicit fihm(QWidget *parent = 0);
    ~fihm();

private:
    Ui::fihm *ui;
    QSqlDatabase bdd;
    QTimer timer;
    QSqlQuery query;
    QString hydrovalue;
public slots:
    void timerslot();
};

#endif // FIHM_H
```

Le .h va permettre la déclaration des divers variables et fonction utiliser par le programme (.cpp).

La variable de la base de données est dans le .h

La classe QSqlDatabase représente la connexion à la base de données.

Elle fournit une interface pour accéder à une base de données via une connexion.

Le QTimer va permettre de fournir une minuterie répétitive qui va permettre de temporiser les données qui vont être envoyés de la base de données.

Pour cela il crée un timer qui va émettre un signal à des intervalles constants.

Le QSqlQuery va servir à fournir un moyen d'exécuter et de manipuler les requêtes SQL.

Le QString hydrovalue va permettre de donner la valeur d'hydrométrie.

Voici le code .cpp :

```
#include "fihm.h"
#include "ui_fihm.h"
#include <QTimer>

fihm::fihm(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::fihm)
{
    ui->setupUi(this);
    timer.start(1000);
    bdd = QSqlDatabase::addDatabase("QMYSQL");
    bdd.setHostName("localhost"); // IP DB JOEY
    bdd.setDatabaseName("maristovo");
    bdd.setUserName("root");
    bdd.setPassword("admin");
    connect(&timer, SIGNAL(timeout()), this, SLOT(timerslot()));
}

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

void fihm::timerslot()
{
    query.exec("SELECT * FROM volet");
    while (query.next())
    {
        hydrovalue = query.value(1).toString();
    }
    this->ui->lblVolet->setText(hydrovalue);
}
```

Le .cpp est ce qui contient mon programme.

La configuration de la base de données est dans le .cpp

Le timer émet un signal toutes les 1000 ms

Le QTimer étant configuré avec un tick toutes les 1 seconde une requête sera effectuée afin d'afficher la donnée dans un label.

La configuration représente l'adresse IP de la base de données, le nom de la base de données, le nom d'utilisateur et le mot de passe.

Le Query va fournir les requêtes SQL et va donc récupérer les données de la pluie. Donc le but du programme sera de tester la connectivité entre la base de données et le logiciel créé sous Qt permettant de récupérer la donnée d'hydrométrie et ainsi de l'afficher dans un GUI.

Pour ce faire j'ai utilisé une librairie Qt s'appelant QSql effectuant la connexion et la manipulation de la bdd grâce à des requêtes SQL.

6.7 Création d'une base de données test :

Pour pouvoir voir si mon programme récupère des données sur une base de données j'ai créé une base de données test sous ma raspberry.

J'ai rencontré des difficultés pour la faire sous phpmyadmin donc j'ai utilisé le mode graphique sous le terminal.

J'ai donc installé Apache 2 et MySQL.

J'ai créé une table MARI_Volet et j'ai rentré la valeur 15.

Dans le .cpp de mon programme j'ai rentré l'adresse 127.0.0.1

Quand j'ai compilé le label « attente » a donné le nombre 15.

Donc mon programme récupère bien les données de la base de données.

Grâce à cela par la suite je pourrai récupérer les données pour ma partie dans la base de données de l'IR4.

L'heure demandée ou les données de la pluie.

Voici l'exemple de la database :

```
Database
+-----+
| information_schema |
| Maristovoir2       |
| maristovo          |
| maristovoIR2       |
| mysql              |
| performance_schema |
| phpmyadmin         |
+-----+
7 rows in set (0.01 sec)
```

On peut voir le nom de ma base de données.

Pour faire un texte de la connectivité j'ai :

```
if ( ! bdd.open() )
```

```
ui->statusBar->showMessage(bdd.lastError().text()) ;
```

Cela permet de tester la connexion de la base de données.

6.8 Communication I2C :

La communication I2C à pour but de relier facilement un microprocesseur et différent circuits.

Pour communiquer entre la raspberry pi et le volet roulant je dois utiliser la communication I2C.

La raspberry pi marche en 3.3 volts et mon capteur de pluie marche avec une carte Arduino.

Une Arduino marche en 5 volts.

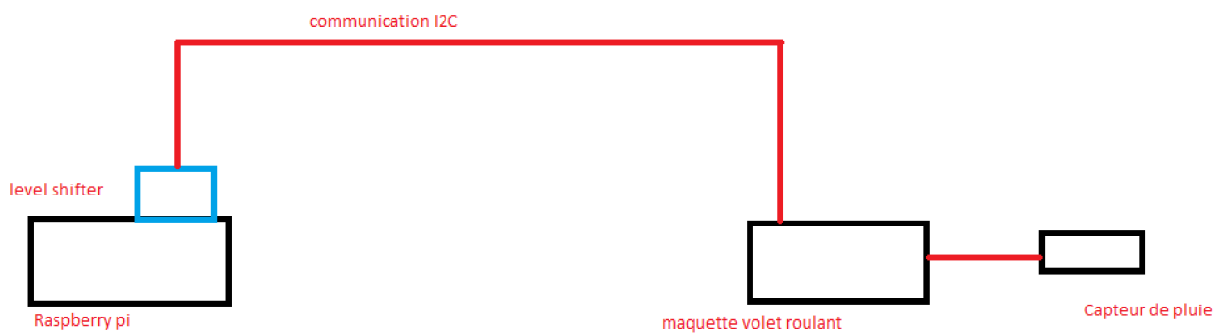
C'est pour cela qu'un level shifter nous a été donné pour pouvoir établir une communication entre la raspberry et une Arduino.

Il m'a été donné par l'EC qui travaille sur toute la partie électronique.



Ici on peut voir le level shifter branché sur la raspberry pi.

Voici un schéma de la communication I2C d'ont j'aurai besoin :



Le capteur de pluie et la raspberry pi auront besoin de cette communication pour se connecté avec le volet roulant.

6.9 Programme I2C sous QT-Créator :

Pour pouvoir utiliser cette communication I2C Avec mes camarades de projet nous avons fais sous QT- créator un programme I2C.

```
#ifndef CI2C_H
#define CI2C_H

#include <QObject>
#include <QDebug>
#include <stdio.h>
#include <unistd.h> //Needed for I2C port
#include <fcntl.h> //Needed for I2C port
#include <sys/ioctl.h> //Needed for I2C port
#include <linux/i2c-dev.h> //Needed for I2C port

class CI2c : public QObject
{
    Q_OBJECT

public:
    // creation destruction de l'objet
    static CI2c *getInstance(QObject *parent = 0, char no = '1');
    static void freeInstance();
    int lire(unsigned char addr, unsigned char *buffer, int lg);
    int ecrire(unsigned char addr, unsigned char *buffer, int lg);
    int init();
    int getNbLink();

private:
    explicit CI2c(QObject *parent = 0, char noBus = '1');
    int mAddr; // Adresse du composant I2C
    char mNoBus; // No d'accès au fichier /dev
    int mFileI2c; // descripteur du fichier i2c
    int mNbLink;
    static CI2c *mSingleton;
};

#endif // CI2C_H
```

On peut voir la déclaration des variables et des données.

En public on peut voir :

Il y a la création et la destruction de l'objet.

En priver on peut voir :

L'adresse du composant I2C

Le descripteur du fichier I2C

et l'accès au fichier demandé.

Ce .h va donc faire la création de l'objet et donnée l'accès à la librairie I2C.

Le .cpp de ce code va permettre de lire, écrire et initialiser.

Il va pouvoir faire la communication entre la raspberry et le volet roulant.

```
#include "ci2c.h"

CI2c::CI2c(QObject *parent, char noBus) :
    QObject(parent)
{
    mNoBus = noBus;
    mNbLink=0;
} // constructeur

CI2c * CI2c::mSingleton = NULL;

int CI2c::lire(unsigned char addr, unsigned char *buffer, int lg)
{
    if(ioctl(mFileI2c, I2C_SLAVE, addr)!=0) { // Règle le driver I2C sur l'adresse.
        qDebug("Erreur ioctl acces au bus I2C");
        return -1;
    } // if ioctl
    int nb=read(mFileI2c, buffer, lg);
    return nb;
} // lire

int CI2c::ecrire(unsigned char addr, unsigned char *buffer, int lg)
{
    if(ioctl(mFileI2c, I2C_SLAVE, addr)!=0) { // Règle le driver I2C sur l'adresse.
        qDebug("Erreur ioctl acces au bus I2C");
        return -1;
    } // if ioctl
    int nb=write(mFileI2c, buffer, lg);
    return nb;
} // ecrire

int CI2c::init()
{

```

Le programme commence avec le constructeur pour faire les objets en fonction des classes.

Cette partie va régler le driver I2C sur l'adresse qu'il faut.

Le qDebug va permettre de voir s'il y a une erreur pour accéder au bus I2C.

Il retourne -1 s'il y a une erreur.

S'il n'y a pas d'erreur il retourne un nombre.

On peut voir qu'il y a la partie lire et la partie écrire.

Voici L'autre partie du programme I2C .cpp

```
int CI2C::init()
{
    char filename[20];
    qDebug() << mNoBus;
    sprintf(filename, "/dev/i2c-%c", mNoBus);
    if((mFileI2C=open(filename, O_RDWR))==-1) { // ouvre le fichier virtuel d'accès à l'I2C
        qDebug("Erreur ouverture acces au bus I2C");
        return -1;
    } // if open
    return mFileI2C;
} // init

int CI2C::getNbLink()
{
    return mNbLink;
} // getNbLink

CI2C *CI2C::getInstance(QObject *parent, char no)
{
    if (mSingleton == NULL)
    {
        qDebug("L'objet CI2C sera créé !");
        mSingleton = new CI2C(parent, no);
        mSingleton->init();
        mSingleton->mNbLink++;
    }
    else
    {
        qDebug("singleton already created!");
    }
    return mSingleton;
} // getInstance

void CI2C::freeInstance()
{
    if (mSingleton != NULL)
    {
        {
            mSingleton->mNbLink--;
            if (mSingleton->mNbLink==0) {
                close(mSingleton->mFileI2C);
                delete mSingleton;
                mSingleton = NULL;
            } // if mNbLink
        } // if null
    } // freeInstance
}
```

Dans cette partie on peut voir l'ouverture de la communication I2C.

On peut constater que l'objet CI2C est construit.

Grace à l'objet crée est la liaison faite la raspberry pourra communiquer avec le volet roulant et le capteur de pluie.

6.10 Capteur de pluie :

6.10.1 Présentation :

Le volet roulant doit descendre automatiquement quand il y a de la pluie.

C'est là où le capteur de pluie rentre en jeu.

Après plusieurs recherches sur le mode de fonctionnement des détecteurs de pluie basique, le capteur de pluie utilisé pour le projet est complètement différent.

Le capteur de pluie utilisé est une résistance.

La résistance va être reliée à une carte Arduino.

Résistance Utilisée :

La résistance utilisée est une FC-37



Cette résistance marche en quelque mV.

Elle va devoir fournir 3 états :

L'état « pas de pluie »

L'état « Risque de Pluie »

L'état « pluie »

Ces 3 états sont présents en fonction de la pluie.

6.11 Carte Arduino utilisé :

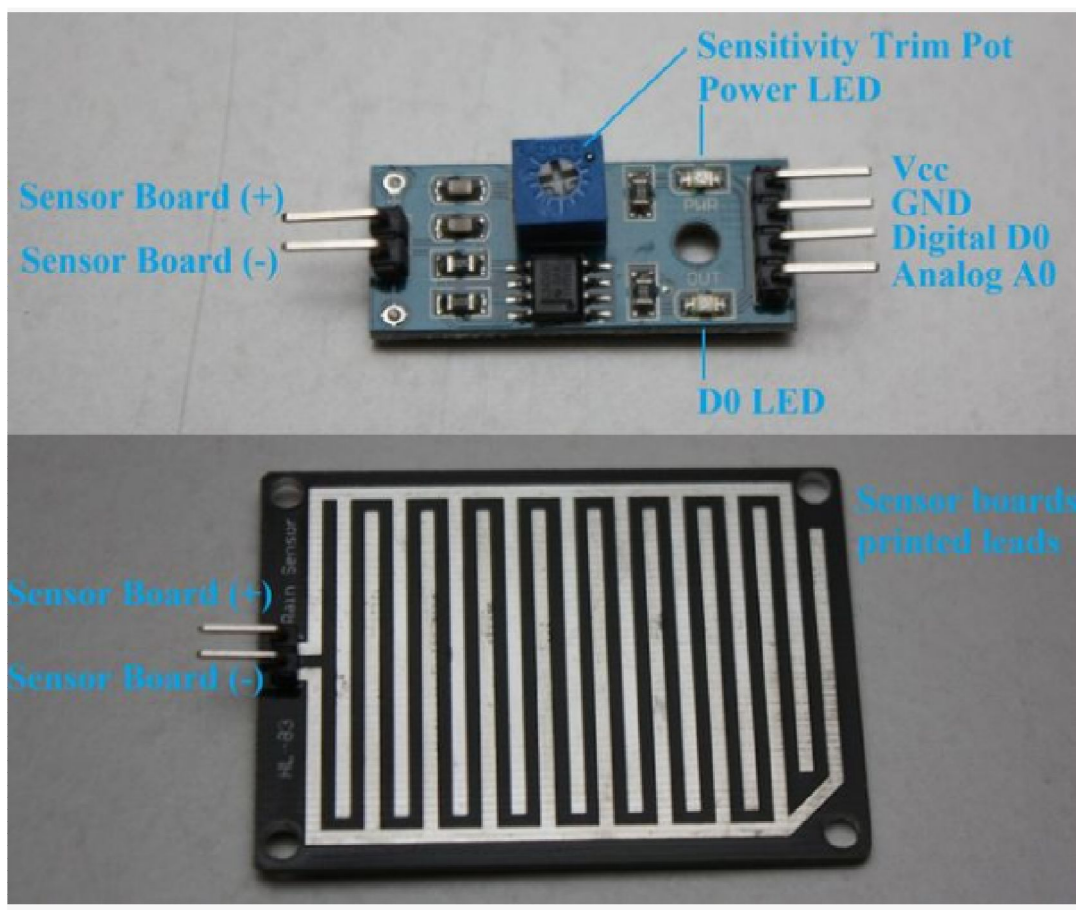
J'utilise une carte Arduino Uno.

Cette carte Arduino Uno est basée sur un ATmega328 cadencé à 16 MHz.

C'est la plus récente et la plus économique carte à microcontrôleur d'Arduino. Des connecteurs situés sur les bords extérieurs du circuit imprimé permettent d'enficher une série de modules complémentaires.

la carte Arduino marche en 5 Volts et la résistance en quelque mvolts , il ne pouvait pas être relié directement .

C'est pour cela qu'un amplificateur a été mis entre eux.



On peut voir que sur l'amplificateur il y a plusieurs Pins :

A0 : Analog output

D0 : Digital output

GND : Ground

VCC : Positive voltage (input: 5v for analog 3.3v for Digital.)

loop Pins :

+ : Sensor board hookup A

- : Sensor board hookup B

Puis la carte de capteur positive et la carte de capteur négative.

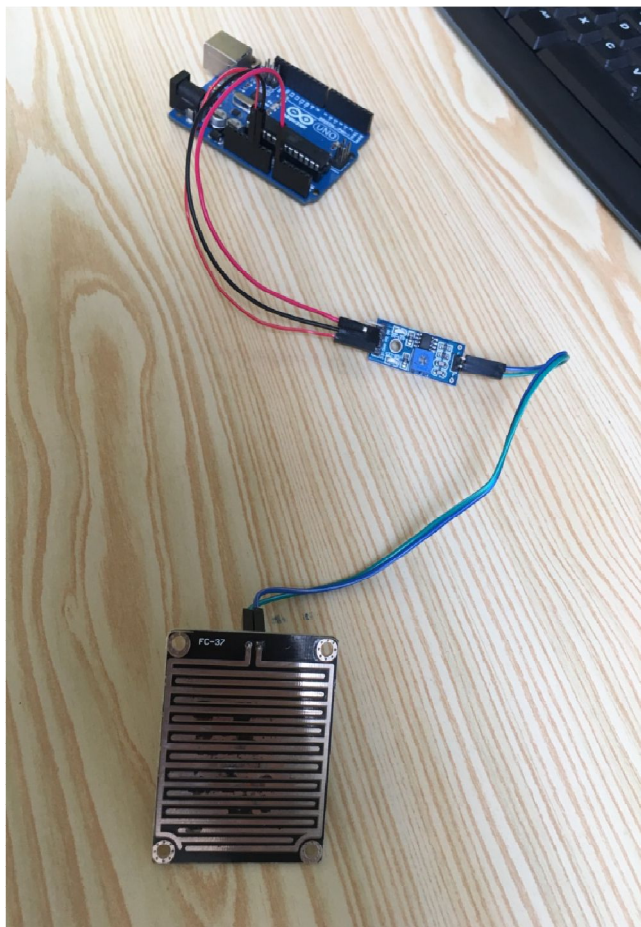
Le capteur de pluie va pouvoir détecter la pluie pas comme un capteur d'humidité qui ne va pas pouvoir.

Comment marche-t-il ?

Le capteur de pluie détecte de l'eau qui va compléter le circuit imprimés.

Le capteur va agir comme une résistance variable qui va changer à partir de 100 Kohms quand il sera mouillé.

Quand il sera sec il sera à 2 Mohms.



Voici la manière de relier la carte Arduino au capteur :

On peut voir que la carte Arduino est Relié par 3 fils à l'amplificateur.

L'amplificateur est lui relié à la Résistance.

Les pins utilisé sont : le 5v de l'Arduino relié au vcc de L'amplificateur.

Le GND de l'Arduino est relié au GND de l'amplificateur.

Le A0 de la carte Arduino est relié au A0 de l'amplificateur

Puis l'amplificateur est relié au capteur.

6.12 Programme capteur de pluie :

Pour pouvoir faire fonctionner le capteur de pluie j'ai du télécharger le logiciel Arduino et faire un programme pour qu'il détecte la pluie.

Voici le programme réalisé :

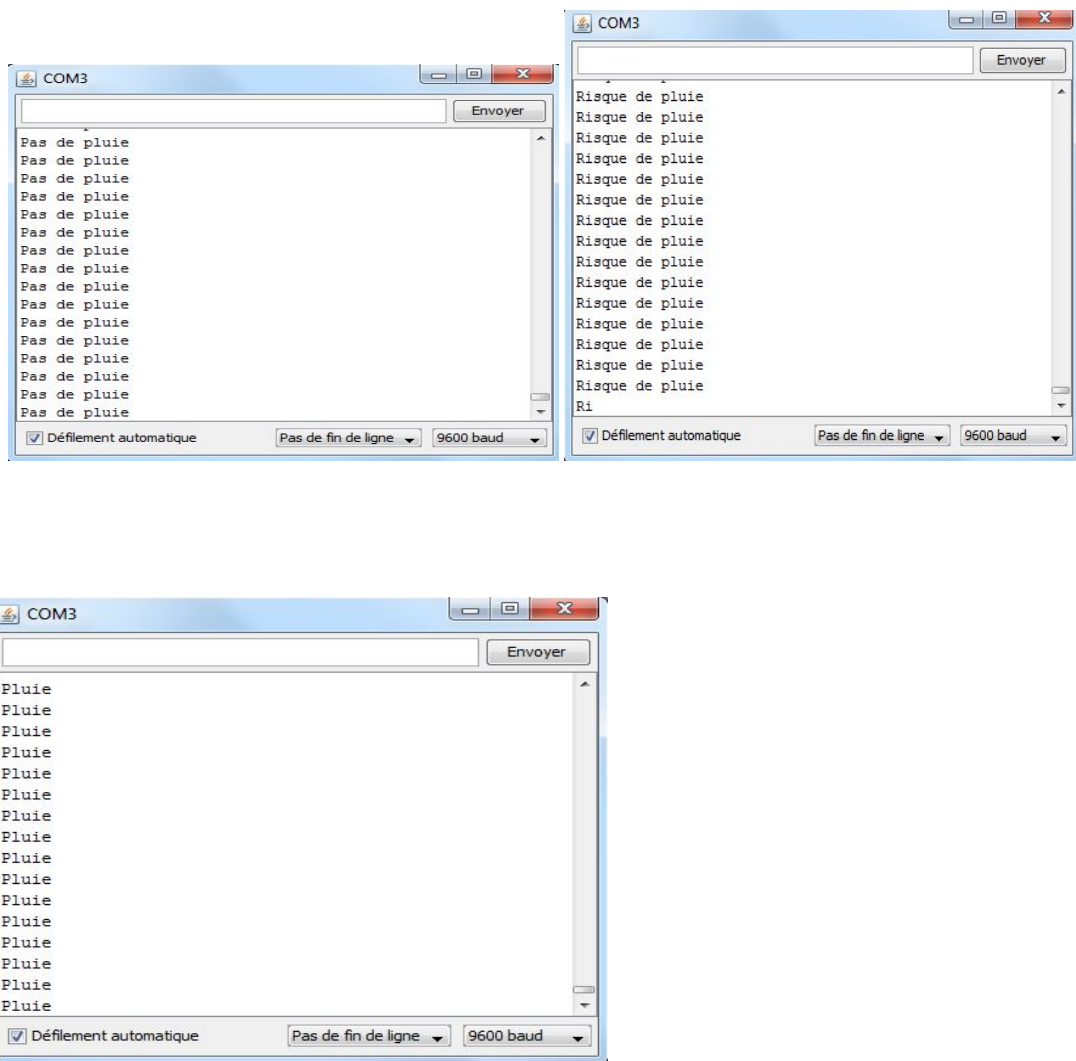
```
const int sensorMin = 0;    // sensor minimum
const int sensorMax = 1024; // sensor maximum

void setup() {
  Serial.begin(9600);
  pinMode(2, OUTPUT); // red led
  pinMode(3, OUTPUT); // yellow led
  pinMode(4, OUTPUT); // green led
}

void loop() {
  int sensorReading = analogRead(A0); // read the sensor on analog A0
  int range = map(sensorReading, sensorMin, sensorMax, 0, 3); // map the sensor range (four options)
  switch (range) { // range value
    case 0: // Sensor getting wet
      Serial.println("Pluie");
      digitalWrite(2, HIGH);
      digitalWrite(3, LOW);
      digitalWrite(4, LOW);
      break;
    case 1: // Sensor getting wet
      Serial.println("Risque de pluie");
      digitalWrite(2, LOW);
      digitalWrite(3, HIGH);
      digitalWrite(4, LOW);
      break;
    case 2: // Sensor dry - To shut this up delete the " Serial.println("Not Raining"); " below.
      Serial.println("Pas de pluie");
      digitalWrite(2, LOW);
      digitalWrite(3, LOW);
      digitalWrite(4, HIGH);
      break;
  }
  delay(1);
}
```

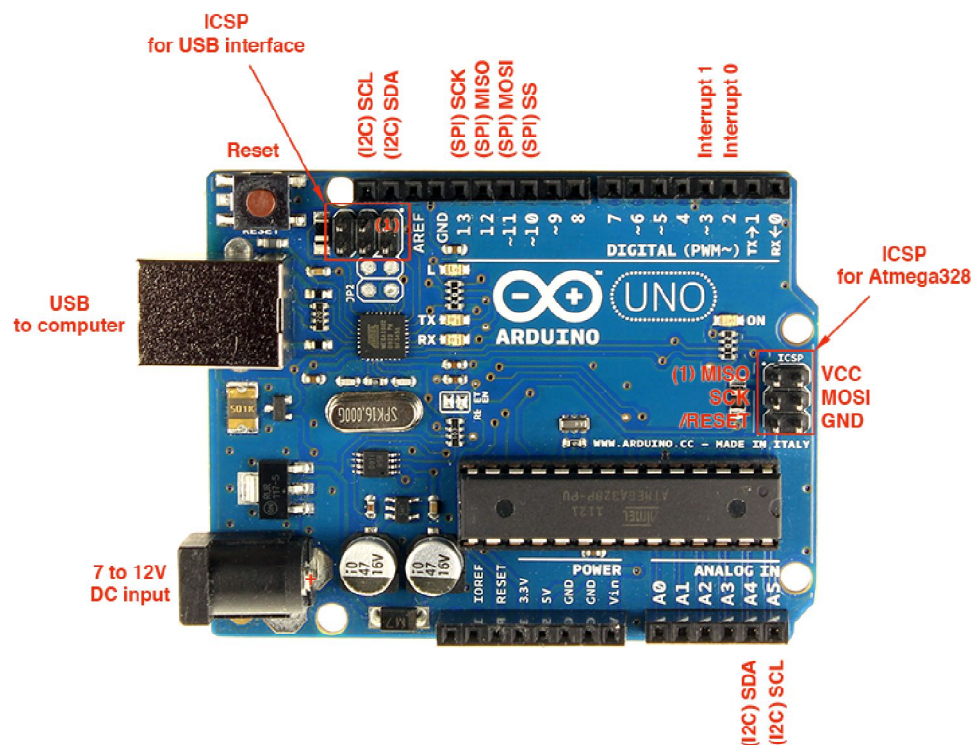
Ce programme va permettre de voir si il ya de la pluie ou non en fonction de l'eau mise sur le capteur.

On peut voir les 3 différents niveaux donnés par l'Arduino :



Pour pouvoir communiquer avec la raspberry L'Arduino va être relié au volet roulant par la communication I2C.

Les ports I2C pour l'Arduino uno sont :



Les ports I2C sont donc : A4 et A5

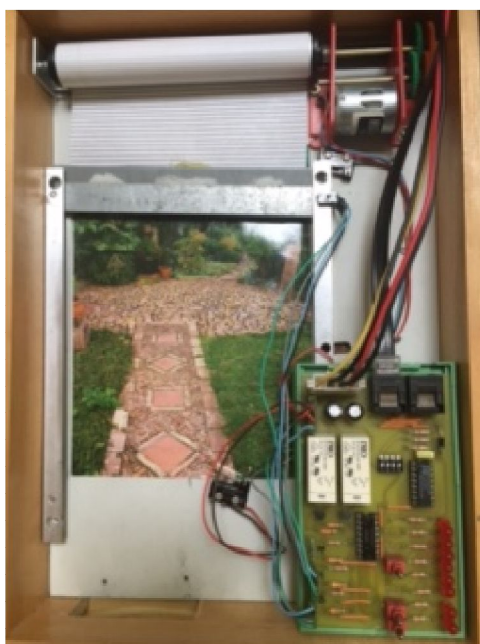
6.13 Volet roulant :

J'ai rencontré beaucoup de difficulté à faire monter et descendre le volet roulant.

Pour cela j'ai collaboré avec mes camarades IR1 et IR3.

Pour cela j'ai regardé comment la raspberry pouvait se connecter à la maquette.

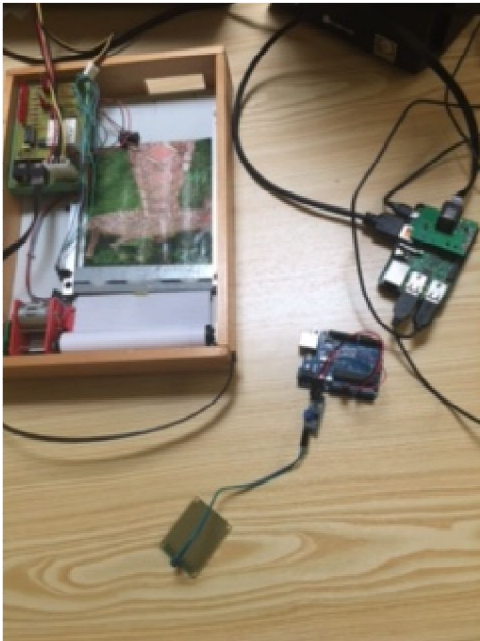
Pour simuler le volet roulant j'ai une maquette.



Ceci représente la maquette du volet roulant.

On peut voir qu'il doit être simulé par un programme.

Voici le câblage qui devra être fait :



Pour tous connecté entre eux il faut se servir de la connexion Entre eux.

Le volet roulant va être en connexion avec la raspberry par la Communication I2C grâce au level shifter.

Puis pour que mon capteur soit connecté il va falloir que je

Connecte les port I2C donc A4 et A5 à ma raspberry.

Cela fera une connexion établie pour le bon fonctionnement du capteur et de la raspberry.

6.14 Programme sous QT-Creator pour le volet roulant :

```

#ifndef VOLET_ROULANT_H
#define VOLET_ROULANT_H

#include <QThread>
#include <QMutex>
#include "ci2c.h"

class CI2c;

class Volet_Roulant: public QThread
{
    Q_OBJECT
public:
    explicit Volet_Roulant(QObject *parent = 0);
    ~Volet_Roulant();
    void run();
    void Monter();
    void Descendre();
    void Stop();
    void SetMonter(bool Monter);
    void SetDescendre(bool Descendre);
    void SetModeManuel(bool ModeManu);
    void SetModeAutomatique(bool ModeAuto);
    void SetValeurPluie(QString Pluie);
    void SetActualiser(QString ParametreOuverture, QString ParametreFermeture);
signals:

public slots:

private:
    bool FinDeCourseHaut;
    bool FinDeCourseBas;
    bool ModeManuel;
    bool ModeAutomatique;
    bool MonterVolet;
    bool DescendreVolet;
    double SeuilOuverture;
    double SeuilFermeture;
    unsigned char valeur,test;

    double ValeurPluie;

    CI2c *BusI2c;
    QMutex *Mutex;
};

#endif // VOLET_ROULANT_H

```

Dans le .h dans la partie public plusieurs fonctions sont créées.

Ces fonctions vont servir à devoir monter et descendre le volet en mode manuel ou automatique.

Puis de régler les paramètres ouverture et fermeture.

Dans la partie privée on peut voir qu'il y a des variables de type 1 ou 0. Ou le volet monte ou le volet descend.

```
void Volet_roulant::SetModeAutomatique(bool ModeAuto)
{
    ModeAutomatique = ModeAuto;
} //SetModeAuto

void Volet_roulant::SetModeManuel(bool ModeManu)
{
    ModeManuel = ModeManu;
} //SetModeManuel

void Volet_roulant::SetValeurPluie(QString Pluie)
{
    ValeurPluie = Pluie.toDouble();
} //SetValeurPluie

void Volet_roulant::SetMonter(bool Monter)
{
    MonterVolet = Monter;
} //SetMonter

void Volet_roulant::SetDescendre(bool Descendre)
{
    DescendreVolet = Descendre;
} //SetDescendre

void Volet_roulant::SetActualiser(QString ParametreOuverture, QString
ParametreFermeture)
{
    SeuilOuverture = ParametreOuverture.toDouble();
    SeuilFermeture = ParametreFermeture.toDouble();
} //SetActualiser

Volet_roulant::~Volet_roulant()
{
}
```

Dans le .cpp on peut voir que le programme contient toutes les fonctionnalités.

Le mode manuel et le mode automatique sont définis.

Le volet roulant est définie selon la pluie.

Puis la monter et la descente du volet roulant est définie

On peut voir que les seuils d'ouverture et de fermeture sont créés.

(Vois annexe pour tous le .cpp)

6.15 Notion de physique :

6.15.1 I2C :

Ma partie physique a été consacrée au protocole I2C.

Le bus I²C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement trois fils : un signal de données (SDA), un signal d'horloge (SCL), et un signal de référence électrique (masse).

Il s'agit d'une liaison en mode série, ce qui signifie que la vitesse de transfert sera plus faible qu'avec un bus de type parallèle. Le bus I²C permet cependant des échanges à la vitesse de 100 kbits par seconde. Certes, la vitesse de transfert du bus I²C n'est pas fulgurante, mais dans bien des cas, la vitesse n'est pas l'élément prédominant..

Les équipements sont donc câblés sur le bus par le principe du « ET câblé », ce qui veut dire qu'en cas d'émission simultanée de deux équipements, la valeur 0 écrase la valeur 1.

État logique d'un protocole I2C :

- que l'état logique « 0 » ou « LOW » est l'état « dominant »,
- que l'état logique « 1 » ou « HIGH » est l'état « récessif ».

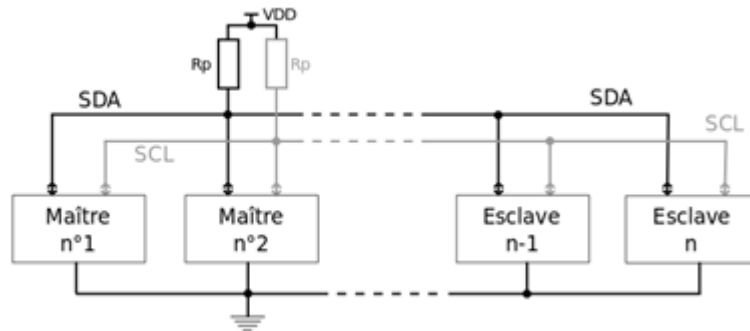
Lorsque le bus n'est pas utilisé, il est au niveau haut

Il existe 5 vitesses de transmission :

- « Standard mode » ≤ 100 kbit/s,
- « Fast mode » ≤ 400 kbit/s,
- « Fast plus mode » ≤ 1 Mbit/s,
- « High-speed mode » $\leq 3,4$ Mbit/s,
- « Ultra-fast mode » ≤ 5 Mbit/s.

Le bus étant synchrone, le maître impose l'horloge via la ligne SCL, il existe des temps minimum à respecter pour les paliers des niveaux « LOW » et « HIGH »

Il utilise le mode maître esclave



La terminologie du bus I2C se fait avec :

Le transmetteur : Le circuit qui envoie la donnée sur le bus

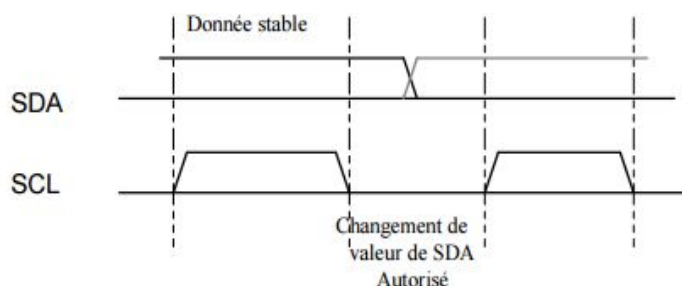
Le receveur : Le circuit qui reçoit la donnée du bus

Le maître : Le circuit qui commence le transfert, génère l'horloge et termine le transfert

L'esclave : Le circuit adressé par le maître

Dans l'I2C il y a le transfert des bits :

La donnée sur la ligne SDA doit être stable pendant que l'horloge est haute sur SCL. La valeur de SDA peut uniquement changer quand SCL est à l'état bas.

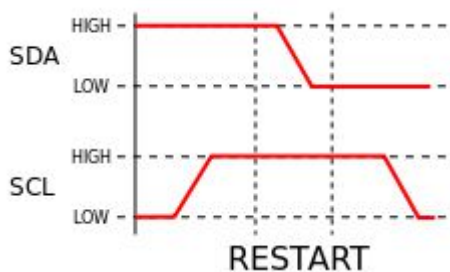


$$R_{Pmin} = \frac{V_{DD} - V_{OLmax}}{I_{OL}}$$

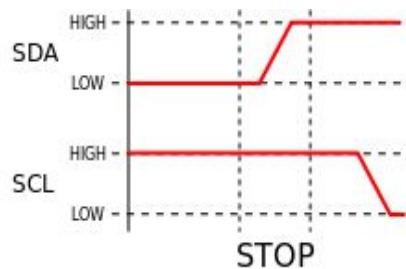
Pour calculer Rp il y a une formule :

Cette formule va calculer la valeur minimale des résistance. Elle est limité par le courant des sorties SDA et SCL quands elles sont à l'état LOW

Voici la condition Restart



Puis conditions Stop



6.16 Partie : Capteur de pluie :

Le capteur de pluie est une résistance qui a comme formule : $U = R \cdot I$

Il est relié par un amplificateur.

Cet amplificateur va pouvoir faire la connexion entre le capteur et l'Arduino donc convertir leurs données

6.17 Conclusion

6.17.1 Ressentie du projet :

Ce projet a été très intéressant pour moi.

J'ai pu apprendre à gérer un projet avec mes camarades et donc pouvoir travailler en groupe.

Pour moi travailler en groupe est très important car ça rassemble plusieurs savoirs.

Malgré certaines difficultés tous le groupe a réussi à collaborer ensemble et à pouvoir travailler en harmonie.

Tous le monde a eu son rôle car ce projet a été découpé en 5 parties chaque personne avait une tâche attribuée.

6.17.2 Difficultés rencontrées :

J'ai rencontré plusieurs difficultés surtout dans le domaine de la programmation.

Le plus dur a été de comprendre comment relier le capteur de pluie au volant roulant.

Après plusieurs aides et de l'entraide la difficulté est devenue plus accessible.

La programmation de l'I2C a été très compliquée.

C'est pour cela que tous le groupe s'est réuni pour pouvoir y travailler.

6.17.3 Pour le moment :

Pour le moment ma partie du projet n'es pas fini car il me reste à relié le capteur de pluie à la raspberry par la communication I2C.

Puis de faire en sorte que le volet ce ferme par rapport au horaire donné à la base de données

Après il nous restera plus qu'a assembler nos programme et à faire tous fonctionner selon les contraintes demandé.



6.18 Annexes programme .cpp du volet roulant :

```

#include "volet_roulant.h"

#define Ad8574 0x20    //adresse composant

#define DESCENDRE 0x80 //descente volet : bit 7 à 0
#define MONTER 0x40   //montée volet: bit 6 à 0
#define HAUT 0x01     //capteur haut: bit 1 à 0
#define BAS 0x02      //capteur bas: bit 2 à 0
#define DEPART 0x08   //BP bit 3 à 0
#define CONFIG 0x3f   //bit à 1 : entrée

Volet_roulant::Volet_roulant (QObject *parent) :
    QThread(parent)
{
    BusI2c = CI2c::getInstance();
    valeur= CONFIG | DESCENDRE | MONTER; //Préparation Entrées et
    if((BusI2c->ecrire(Ad8574,&valeur,1))!=0)
    {
        qDebug("Bug");
    }//if
}

void Volet_roulant::run()
{
    while(1)
    {
        while(ModeAutomatique == true)
        {
            if(SeuilOuverture > ValeurPluie)
            {
                // while(1)
                // {
                //     Monter();
                // }//while
            }//if

            if(SeuilFermeture < ValeurPluie)
            {
                // BusI2c->lire(Ad8574,&valeur,1);
                while(valeur == BAS)
                {
                    Descendre ();
                }//while
            }//if

            sleep(1);
        }//while

        while(ModeManuel== true)
        {
            if (MonterVolet == true)
            {
                Monter();
            }//if
            if (DescendreVolet == true)
            {
                Descendre();
            }
        }
    }
}

```

```

        } //if
    } //while

} //while

} //run

void Volet roulant::SetModeAutomatique(bool ModeAuto)
{
    ModeAutomatique = ModeAuto;
} //SetModeAuto

void Volet_roulant::SetModeManuel(bool ModeManu)
{
    ModeManuel = ModeManu;
} //SetModeManuel

void Volet_roulant::SetValeurPluie(QString Pluie)
{
    ValeurPluie = Pluie.toDouble();
} //SetValeurPluie

void Volet_roulant::SetMonter(bool Monter)
{
    MonterVolet = Monter;
} //SetMonter

void Volet_roulant::SetDescendre(bool Descendre)
{
    DescendreVolet = Descendre;
} //SetDescendre

void Volet_roulant::SetActualiser(QString ParametreOuverture, QString
ParametreFermeture)
{
    SeuilOuverture = ParametreOuverture.toDouble();
    SeuilFermeture = ParametreFermeture.toDouble();
} //SetActualiser

Volet_roulant::~Volet_roulant()
{
    BusI2c->freeInstance();
} //Destructeur

void Volet_roulant::Monter()
{
    valeur= CONFIG | DESCENDRE | MONTER; //Préparation Entrées et Arrêt Sorties
    valeur= valeur & ~MONTER;

    if(BusI2c->ecrire(Ad8574,&valeur,1)!=0)
    {
        qDebug("Erreur Montée !");
    } //if

    do
    {
        if(BusI2c->lire(Ad8574,&valeur,1)!=0)
        {
            qDebug("Erreur PCF8574P");
            sleep(1);
            test=(valeur & HAUT); //Sert au debugage
        } //if
    } //do while
    while(valeur & HAUT);
}

```

```

    valeur = valeur | MONTER | CONFIG; //Arrêt sans toucher aux entrées
    if((BusI2c->ecrire(Ad8574,&valeur,1))!=0)
    {
        qDebug("Problème Arrêt !");
    }//if
    sleep(1);
} //Monter

void Volet_roulant::Descendre()
{
    valeur= valeur & ~DESCENDRE;
    if(BusI2c->ecrire(Ad8574,&valeur,1)!=0)
    {
        qDebug("Erreur descendre !");
    }//if
    do
    {
        if(BusI2c->lire(Ad8574,&valeur,1)!=0)
        {
            qDebug("Erreur PCF8574P");
            sleep(1);
            test=(valeur & HAUT); //Sert au debugage
        }//if
    }//do while
    while(valeur & HAUT);
    valeur = valeur | DESCENDRE | CONFIG; //Arrêt sans toucher aux entrées
    if((BusI2c->ecrire(Ad8574,&valeur,1))!=0)
    {
        qDebug("Problème Arrêt !");
    }//if
    sleep(1);
} //Descendre

void Volet_roulant::Stop()
{
    valeur= CONFIG | DESCENDRE | MONTER; //Préparation Entrées et Arrêt Sorties
} //Stop

```

7 Partie personnelle (IR3)

Pinna Océane

1. Introduction



Dans l'objectif de mon projet je dois contrôler un Store banne grâce à un détecteur de vent et aussi une raspberry Pi qui va nous permettre de programmer sur celle-ci.

Mes tâches à réaliser sont :

Installation de : Qt, MySQL, i2c-dev

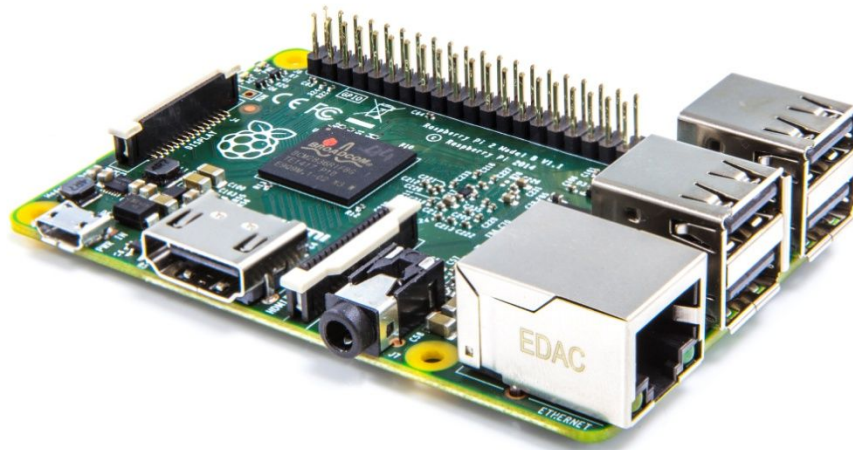
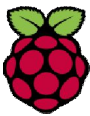
Mise en œuvre : Communication QSerial Port

- Communication I2c
- Acquisition de trames météo

Réalisation : Pilotage du store banne selon les consignes de la base de données et du vent, et simulateur WMR.

Documentation pour l'installation, la mise en service et un dossier de développement

7.1 Présentation de la Raspberry Pi



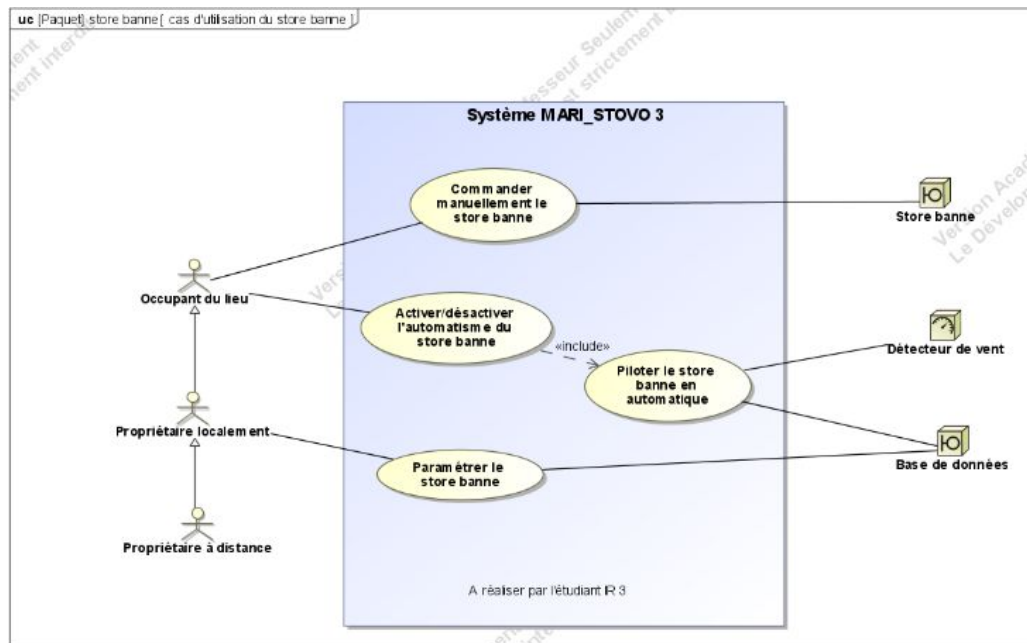
Raspberry Pi est une nano-ordinateur, mono carte conçue par le créateur de jeux vidéo David Braben.

Cet ordinateur, qui a la taille d'une carte de crédit, est destiné à encourager l'apprentissage de la programmation informatique il permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux et des logiciels compatibles. Il est fourni nu (carte mère seule, sans boîtier, alimentation, clavier, souris ni écran) dans l'objectif de diminuer les coûts et de permettre l'utilisation de matériel de récupération.

7.2 Diagrammes

1.2.1 Diagramme cas d'utilisation

En premier voici le diagramme correspondant à la tâche que je dois effectuer :



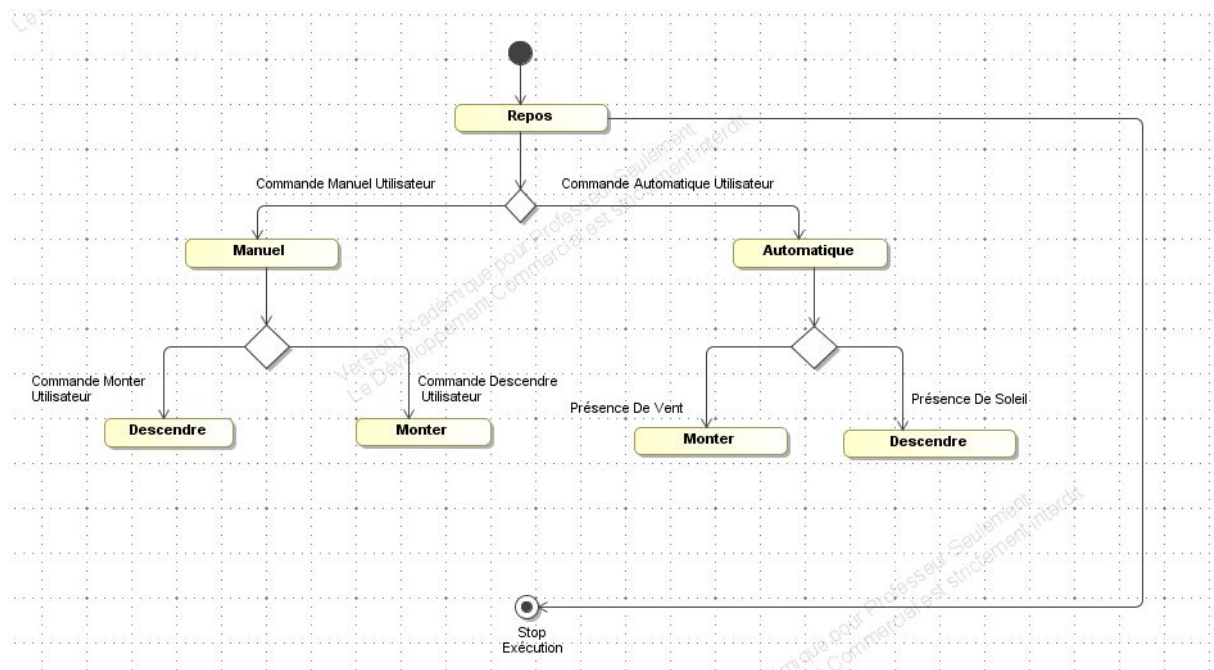
L'occupant du lieu doit pouvoir commander manuellement le store mais aussi automatiquement en paramétrant ce qu'il préfère.

Le store banne doit pouvoir descendre automatiquement grâce à un détecteur de vent qui va pouvoir déterminer quand il y a du vent pour permettre de le monter.

Ensuite le propriétaire locale va pouvoir paramétrer le store banne qui lui même est relié à la base de données.

Ma tâche étant de piloter le store banne selon les consignes de la Base De Données et du vent, ainsi que réaliser un simulateur WMR, pour cela j'ai pus me documenter grâce à la mise en service et un dossier de développement.

1.2.2 Diagramme d'état



Ce diagramme d'état nous permet de visualiser plus facilement comment va réagir le store banne en fonction du mode choisi.

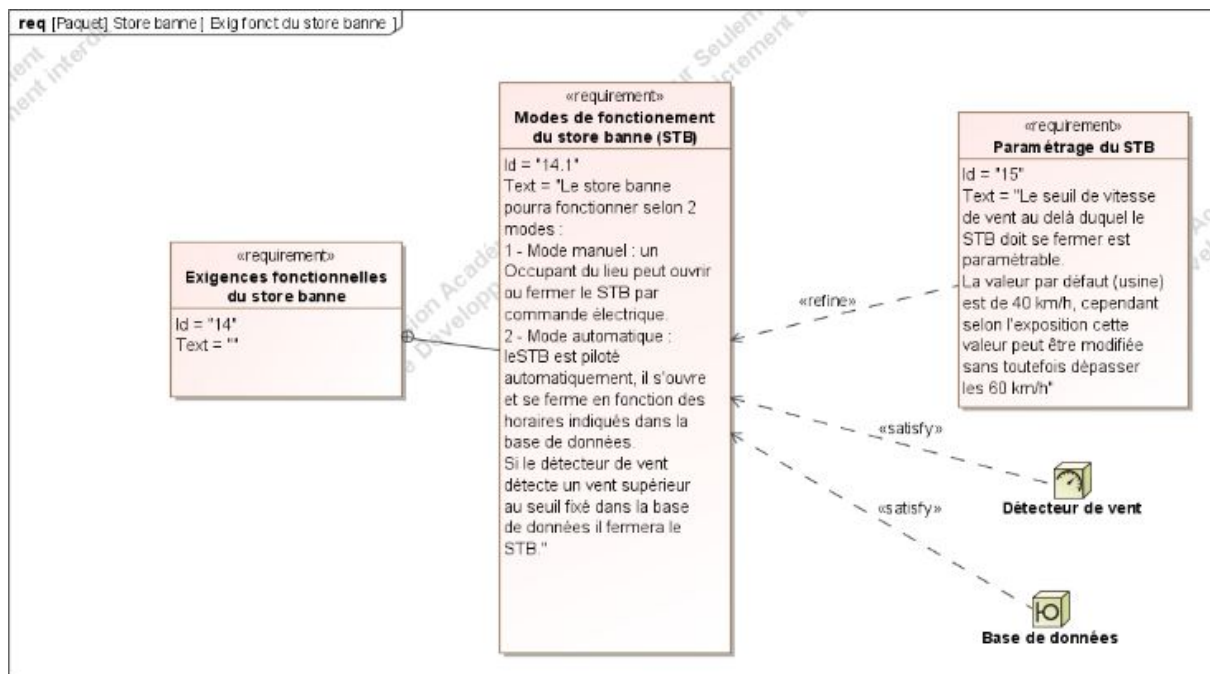
Quand le Store banne est au repos c'est-à-dire qu'il est inactif.

L'utilisateur va devoir choisir entre le mode manuel ou le mode automatique.

Mode Manuel : l'utilisateur va devoir choisir entre descendre ou monter le volet.

Mode Automatique : Le volet va monter si il y a une présence de vent et va descendre s'il y a une présence de soleil.

1.2.3 Diagramme d'exigence

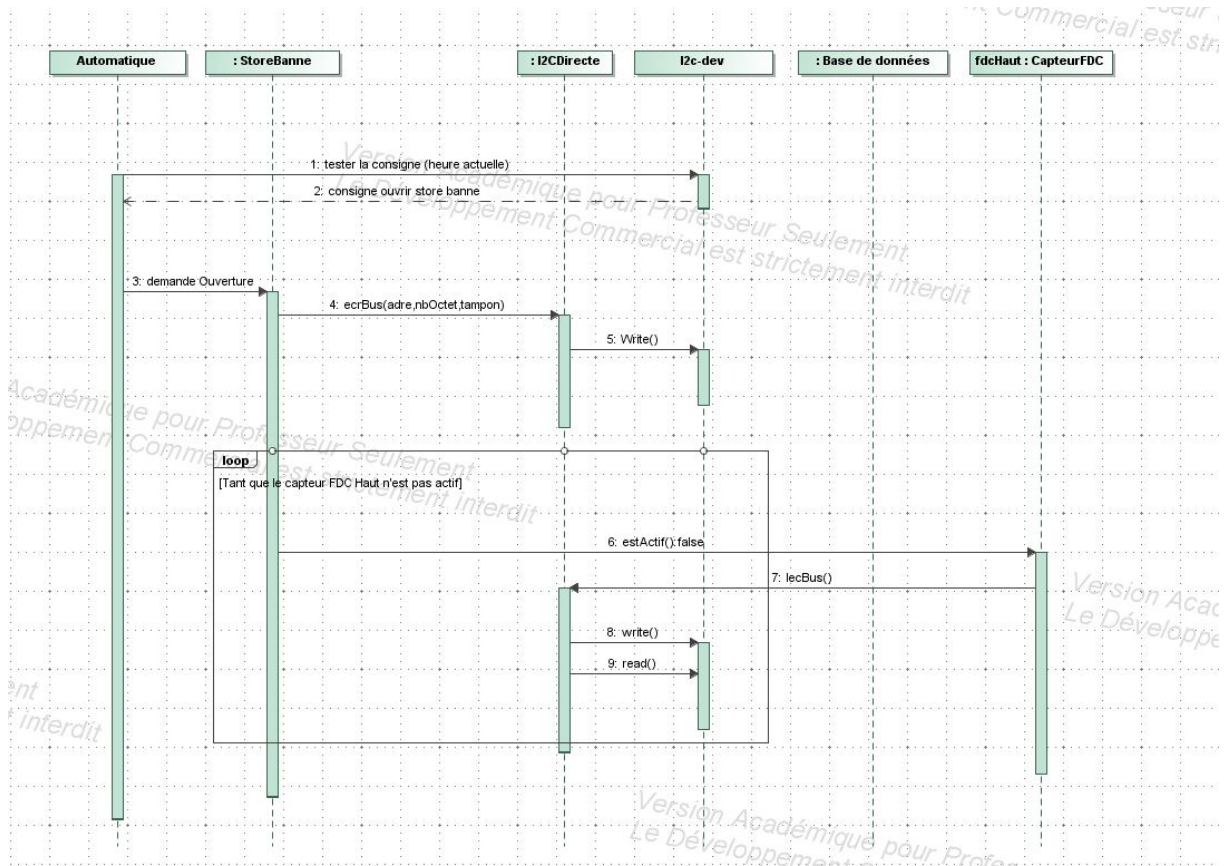


Ce diagramme va nous permettre de visualiser les tâches que je dois accomplir.

Dans un premier temps on peut voir qu'il y a deux modes de fonctionnement : un Mode Manuel qui va permettre à l'occupant du lieu de contrôler électriquement le Store Banne

Un Mode Automatique qui va permettre à l'occupant du lieu de mettre en place des horaires d'ouverture et de fermeture qui seront gérés dans la base de données. Dans ce mode automatique il y a aussi un détecteur de vent où il y aura été fixé un seuil qui dès que le vent dépassera celui-ci le Store banne se fermera automatiquement.

1.2.4 Diagramme de séquences



7.3 Programme

7.3.1 Création de la base de donnée pour permettre de vérifié le bon fonctionnement du programme :

Le programme qui va être vus par la suite va permettre de récupérer les valeurs de la base de données pour pouvoir gérer le Store banne quand il sera en mode automatique.

Tout d'abord nous allons créer une base de données temporaire sur la Raspberry Pi pour tester le bon fonctionnement du programme qui par la suite devra récupérer les vraies valeurs stockées dans la base de données finale.

Pour cela il va falloir installer plusieurs applications :

- Installation Apache2
- Installation MySql
- Installation PhpMyAdmin

Après que celles-ci soient correctement installées il va falloir créer une table : Cette table se nommera "MariStovoIR3 ".

Dans cette table il y aura une valeur nulle qui se retournera dans le programme pour voir que celui-ci fonctionne correctement et qu'il se connecte bien à la base de données.

Pour permettre au programme de se connecter à la base de données il faudra renseigner l'adresse IP de celle-ci, qui sera donc remplacé à la ligne Numéro 12.

```

5  Fihm::Fihm(QWidget *parent) :
6      QMainWindow(parent),
7  ui(new Ui::Fihm)
8  {
9      ui->setupUi(this);
10     timer.start(1000);
11     bdd = QSqlDatabase::addDatabase("QMYSQL");
12     bdd.setHostName("localhost");
13     bdd.setDatabaseName("MariStovoIR3");
14     bdd.setUserName("root");
15     bdd.setPassword("admin");
16     connect(&timer, SIGNAL(timeout()), this, SLOT(timerslot()));
17     etat = false;
18 }

```

7.3.2 .h : fichier d'entête

Cette partie de programme va permettre de déclarer plusieurs variables et fonctions :

```
#ifndef FIHM_H
#define FIHM_H

#include <QMainWindow>
#include <QtSql/qsql.h>
#include <QtSql/SqlDatabase>
#include <QtSql/SqlDriver>
#include <QtSql/SqlQuery>
#include <QTimer>

namespace Ui {
class Fihm;
}

class Fihm : public QMainWindow
{
    Q_OBJECT

public:
    explicit Fihm(QWidget *parent = 0);
    ~Fihm();

private:
    Ui::Fihm *ui;
    QSqlDatabase bdd;
    QTimer timer;
    QSqlQuery query;
    QString windvalue;
public slots:
    void timerslot();

};

#endif // FIHM_H
```

7.3.3 main.cpp

```
#include "fihm.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Fihm w;
    w.show();

    return a.exec();
}
```

Ce programme permet de voir la connectivité entre le logiciel et la Base de données, il devra récupérer les valeurs liées au vent pour l'afficher dans un GUI.

Pour cela j'utilise la librairie QSql sous Qt qui permet la connexion de la Base de données grâce aux requêtes SQL.

J'ai donc utilisé la un QTimer qui est configuré toutes les une seconde avec un tick pour permettre d'afficher les données dans le label.

1.3.4 Protocole I2c

Le protocole I2C est un bus informatique.

Il a été conçu pour les applications de domotique et d'électronique domestique.

Il va permettre de relier un microprocesseur et différents circuits.

Le protocole va pouvoir envoyer les commandes au Store Banne.

Les commandes seront de monter et descendre les volets roulant et récupérer les données du capteur.

Voici le .h

```
#ifndef CI2C_H
#define CI2C_H

#include <QObject>
#include <QDebug>
#include <stdio.h>
#include <unistd.h> //Needed for I2C port
#include <fcntl.h> //Needed for I2C port
#include <sys/ioctl.h> //Needed for I2C port
#include <linux/i2c-dev.h> //Needed for I2C port

class CI2c : public QObject
{
    Q_OBJECT

public:
    // creation destruction de l'objet
    static CI2c *getInstance(QObject *parent = 0, char no = '1');
    static void freeInstance();
    int lire(unsigned char addr, unsigned char *buffer, int lg);
    int ecrire(unsigned char addr, unsigned char *buffer, int lg);
    int init();
    int getNbLink();

private:
    explicit CI2c(QObject *parent = 0, char noBus = '1');
    int mAddr; // Adresse du composant I2C
    char mNoBus; // No d'accès au fichier /dev
    int mFileI2c; // descripteur du fichier i2c
    int mNbLink;
    static CI2c *mSingleton;
};

#endif // CI2C_H
```

Programme .cpp

```

#include "ci2c.h"

CI2c::CI2c(QObject *parent, char noBus) :
    QObject(parent)
{
    mNoBus = noBus;
    mNbLink=0;
} // constructeur

CI2c * CI2c::mSingleton = NULL;

int CI2c::lire(unsigned char addr, unsigned char *buffer, int lg)
{
    if(iocctl(mFileI2c, I2C_SLAVE, addr)!=0) { // Règle le driver I2C sur l'adresse.
        qDebug("Erreur iocctl acces au bus I2C");
        return -1;
    } // if iocctl
    int nb=read(mFileI2c, buffer, lg);
    return nb;
} // lire

int CI2c::ecrire(unsigned char addr, unsigned char *buffer, int lg)
{
    if(iocctl(mFileI2c, I2C_SLAVE, addr)!=0) { // Règle le driver I2C sur l'adresse.
        qDebug("Erreur iocctl acces au bus I2C");
        return -1;
    } // if iocctl
    int nb=write(mFileI2c, buffer, lg);
    return nb;
} // ecrire

int CI2c::init()
{

```

7.3.5 Interface Graphique

L'interface graphique va être constituée de trois boutons Automatique, Manuel, Stop Exécution. D'une plage horaire permettant de choisir l'heure de la monter et/ou de la descente.

Cette interface graphique va permettre à l'utilisateur de choisir le mode Manuel ou le mode Automatique ainsi que de stopper l'exécution complet du store banne.

Le 20 signifie la bonne réception de la base de données .

La case blanche et le bouton va me permettre de simuler le vent .



7.4 Partie physique :



Cet appareil se place sur le côté, à hauteur du store.

Il permet la fermeture de votre store en fonction du réglage de l'anémomètre effectué. En effet, il mesure en permanence la vitesse du vent et dès que celle-ci devient trop élevée, votre store monte automatiquement. Cet appareil empêche ainsi la dégradation de la toile et du mécanisme. Il s'agit d'un système de confort, ne laissez jamais votre store ouvert par grand vent si vous quittez votre domicile pendant quelques heures.

L'anémomètre mesure comme grandeur physique la vitesse ou la force du vent. Ses unités possibles pour cette grandeur physique sont exprimées :

- Kilomètres par heure (km/h)
- Mètres par seconde (m/s)

Les petites hélices de l'anémomètre sont placées sur un axe gradué. Le vent fait tourner les hélices qui tournent de plus en plus vite et montent sur l'axe. Pour mesurer la vitesse du vent, il suffit donc de lire la graduation sur l'axe qui est proportionnelle à la vitesse du vent.

Pour pouvoir lire sur l'anémomètre c'est à dire la vitesse du vent, on regarde où sont situées les hélices sur l'axe gradué. On lit ensuite la graduation.

7.5 Problèmes rencontrés

Dans notre groupe nous avons rencontrés des problèmes pour la communication de la Raspberry Pi et du Store banne.

Des difficultés en programmation ainsi que des difficultés pour simuler le détecteur de vent.

1.6 Conclusion

Ce projet m'as permis de pouvoir approfondir mes connaissances dans l'informatique et plus précisément en programmation, d'avoir une première approche entre les études et les attentes d'une entreprise et d'apprendre à travailler en équipe et d'être à l'écoute de celle-ci.

7.7 Remerciements

Je tenais à remercier mes professeurs ainsi que mon équipe de projet pour l'aide et les solutions qu'ils ont pu m'apporter.

8 Partie personnelle (IR4)

MARTEL Joey

8.1 Présentation du Projet:

Ce Projet consiste en un application permettant a un utilisateur, de pouvoir gérer les différents équipements de son domicile (dans le cas ci présent, ses volets ainsi que ces stores).L'utilisateur aura a sa disposition une application sur son ordinateur, d'où il pourra configurer ses équipements en mode automatique, ou bien les contrôler manuellement. Il aura aussi a sa disposition un site web, permettant de vérifier l'état des différents équipement de son domicile.

8.1.1 Mon Rôle dans le projet:

Mon Rôle dans ce projets est de fournir un système client/serveur entre l'ordinateur de l'utilisateur et la carte Raspberry PI, qui sert de relais entres tout les équipements et la Base de Données.

La Base de Données étant aussi héberger sur la carte Raspberry, Mon travail consiste, en premier lieux, a créer la Base de Données, puis de communiquer avec elle depuis le serveur, ainsi que depuis le client.

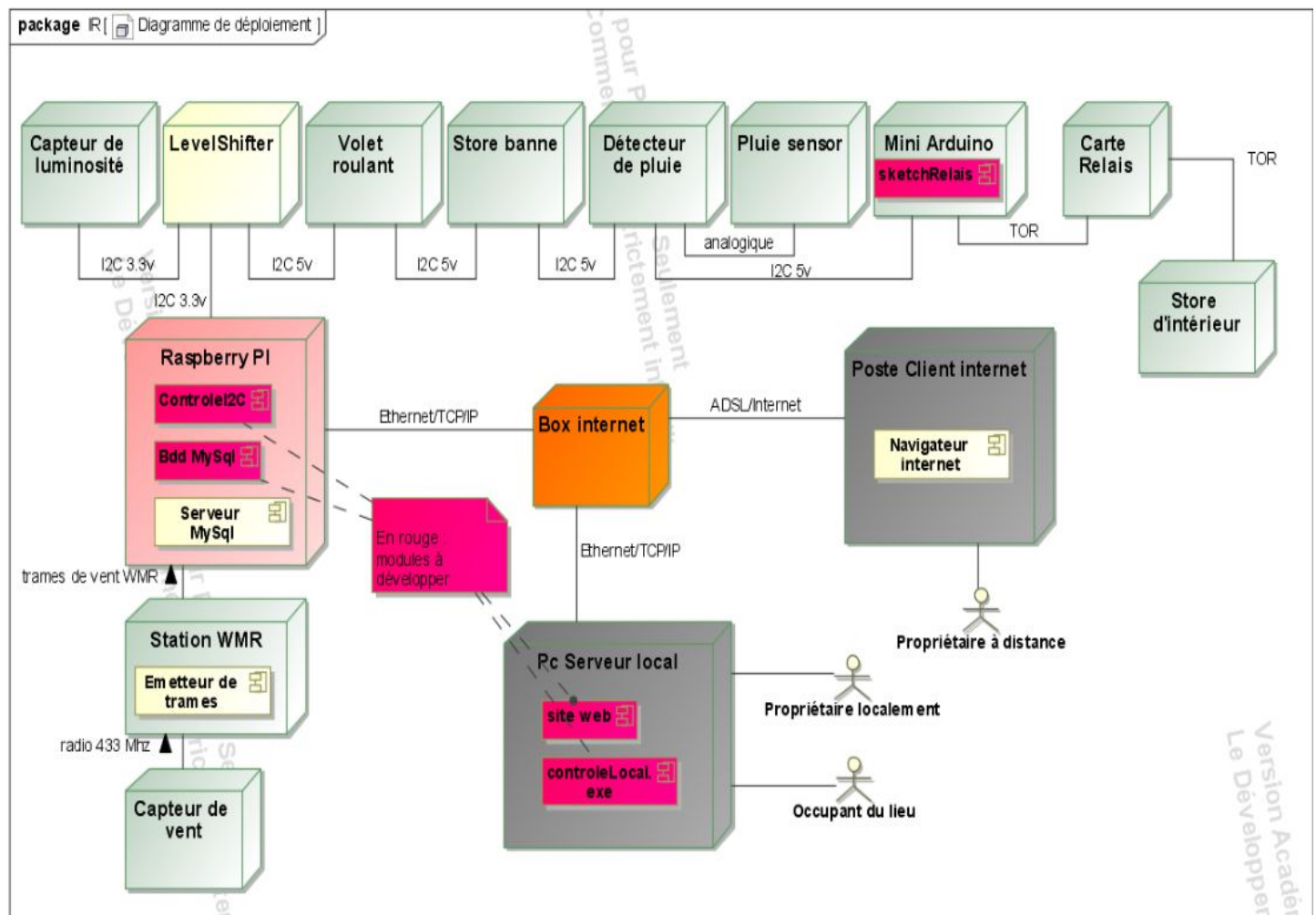
Mon Travail consiste aussi à créer le site web sur lequel l'utilisateur pourra vérifier les équipements de son domicile.

8.2 Diagrammes:

Ce diagramme ci-dessous représente le déploiement de notre projet. Les Parties en Roses représentent les modules à développer.

La Raspberry PI communiquera avec le PC, afin que l'utilisateur puisse envoyer les trames de paramétrages des différents équipements.

La Base de données et stockées sur la Raspberry, et c'est dans la base de donnée que sera stockées les différentes informations, tels que les requêtes de l'utilisateur, ou les relevé des différents capteurs.



8.3 Environnement de Travail:

Pour le développement du Serveur/client TCP, j'ai utilisé QT, un IDE , car c'est l'environnement de développement que je connais le mieux, et de plus , il possède dans ces bibliothèques de base énormément d'aide pour la communication , tant bien au niveau de la communication avec la base de données que pour la communication TCP, tel que QTcpServer.

Pour la Base de données, qui est stocké sur la carte Raspberry pi, j'ai préféré utilisé des logiciels que je connaissais, tel que MySQL, afin de pouvoir gérer la BDD tout en ayant une interface graphique simple d'utilisation.

8.4 Application TCP:

La communication entre l'application de l'exploitant et le serveur se fait grâce à une communication TCP. TCP, étant un protocole dit connecté, permet que le client ou le serveur soit avertis de l'envoi de données, et puisse donc ainsi se préparer à renvoyer une demande, si les données envoyées ont été corrompues ou bien incomplètes.

Cette application se compose donc de deux applications en corrélation, un serveur TCP ainsi qu'un client TCP.

8.4.1 Le Serveur TCP:

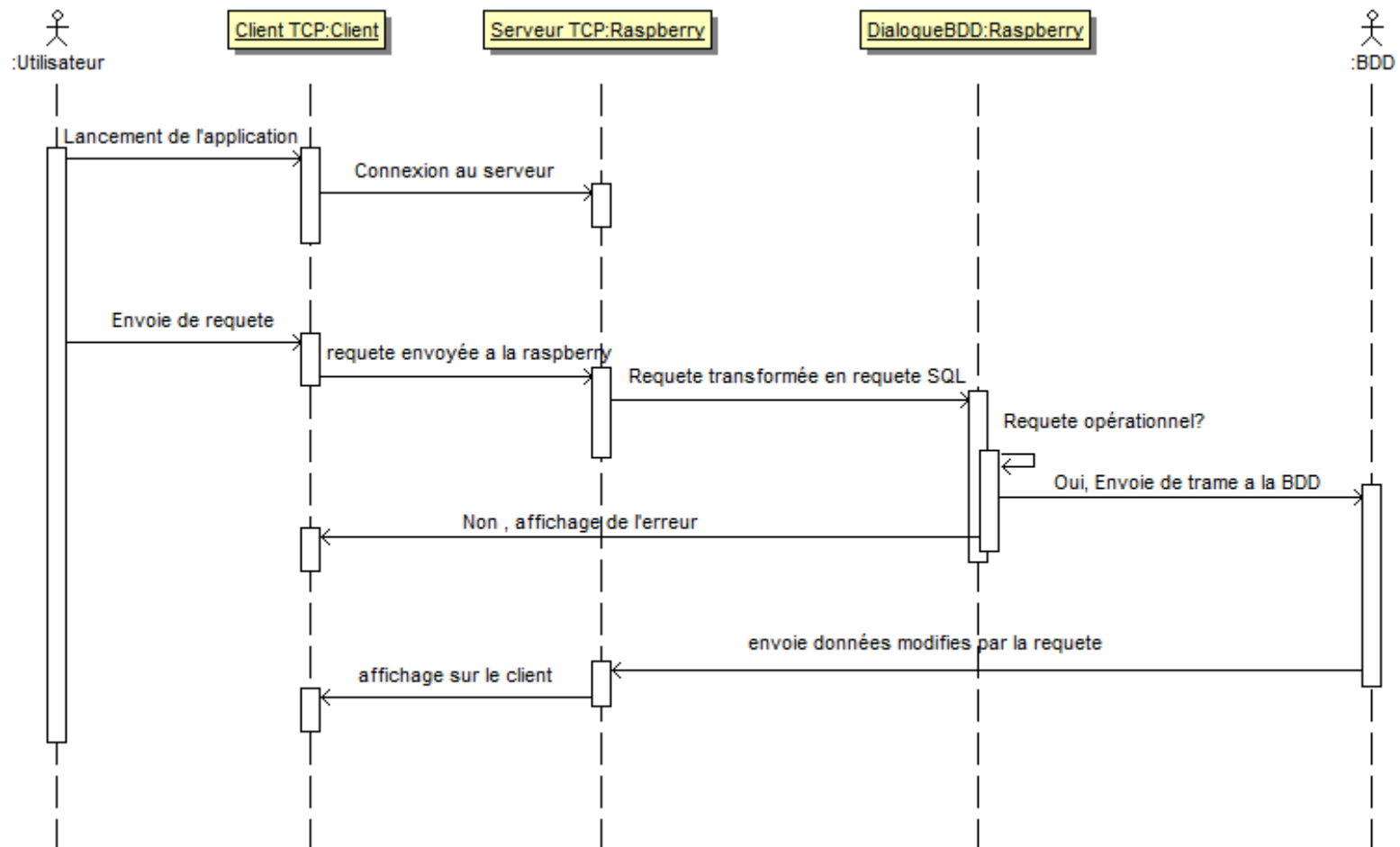
Le serveur TCP est l'application recevant les données du client. Bien qu'il puisse lui-même en envoyer au client, son rôle principal est de servir de connexion entre l'exploitant se trouvant sur son application et la Base de données où sont stockées toutes les données. Il (le serveur) reçoit donc les informations du client grâce au protocole TCP. Il doit ensuite envoyer ces données à la base de données qui les stockera pour différents usages. Le serveur demande aussi des informations à la Base de données tels que les relevés des différents capteurs, afin ensuite de les envoyer à l'application client TCP afin de les afficher sur l'IHM.

8.4.2 Le Client TCP:

Le client TCP sert d'interface à un utilisateur, afin de pouvoir contrôler tout ces équipements. L'utilisateur envoie les données, qui sont ensuite reçues par le serveur puis envoyées dans la base de données. Le client se situe sur un PC distant, ce qui veut dire qu'elle peut être utilisée sur n'importe quel ordinateur tant qu'elle se trouve dans le même réseau que la carte Raspberry.

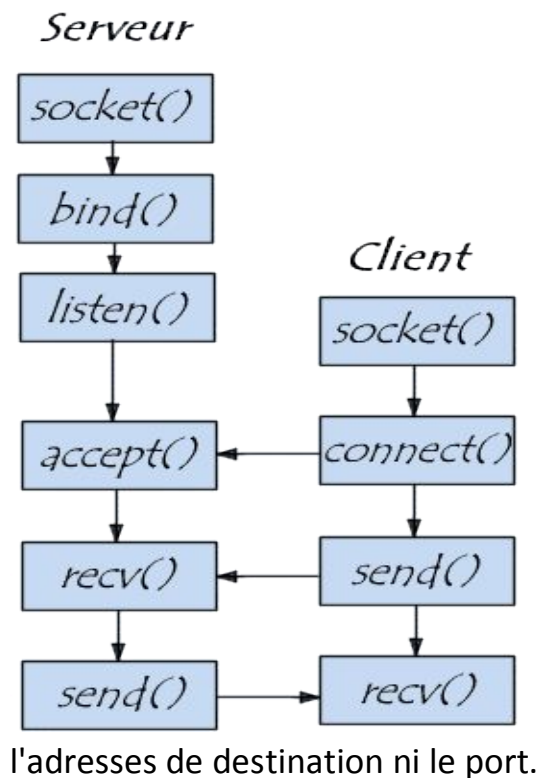
8.4.3 Diagrammes:

Ce diagrammes de séquence représente dans sa globalité le fonctionnement du serveur/client TCP, ainsi que la liaison entre le serveur TCP et l'envoi de données à la base de données, ici appelée par la classe DialogueBDD.



8.5.1 Communication Client/serveur TCP:

La communication entre Client/serveur se fait grâce au classe QT QtcpSocket et QTcpServer.



8.5.2 QTCPSEVEUR :

Cette classe permet de créer un serveur TCP. Son rôle est d'attendre la connexion d'un ou de plusieurs clients, afin des les associer ensuite a une socket afin de pouvoir communiquer avec eux. Le client quant a lui doit pouvoir se connecter au serveur, pour cela il doit connaître son adresse ainsi que son numéro de port. Étant configuré avec le protocole Tcp, protocole dit connecté, le serveur créer une communication dites durables avec le serveur, ce qui signifie qu'ils n'auront pas besoin a chaque requête de renvoyer

l'adresses de destination ni le port.

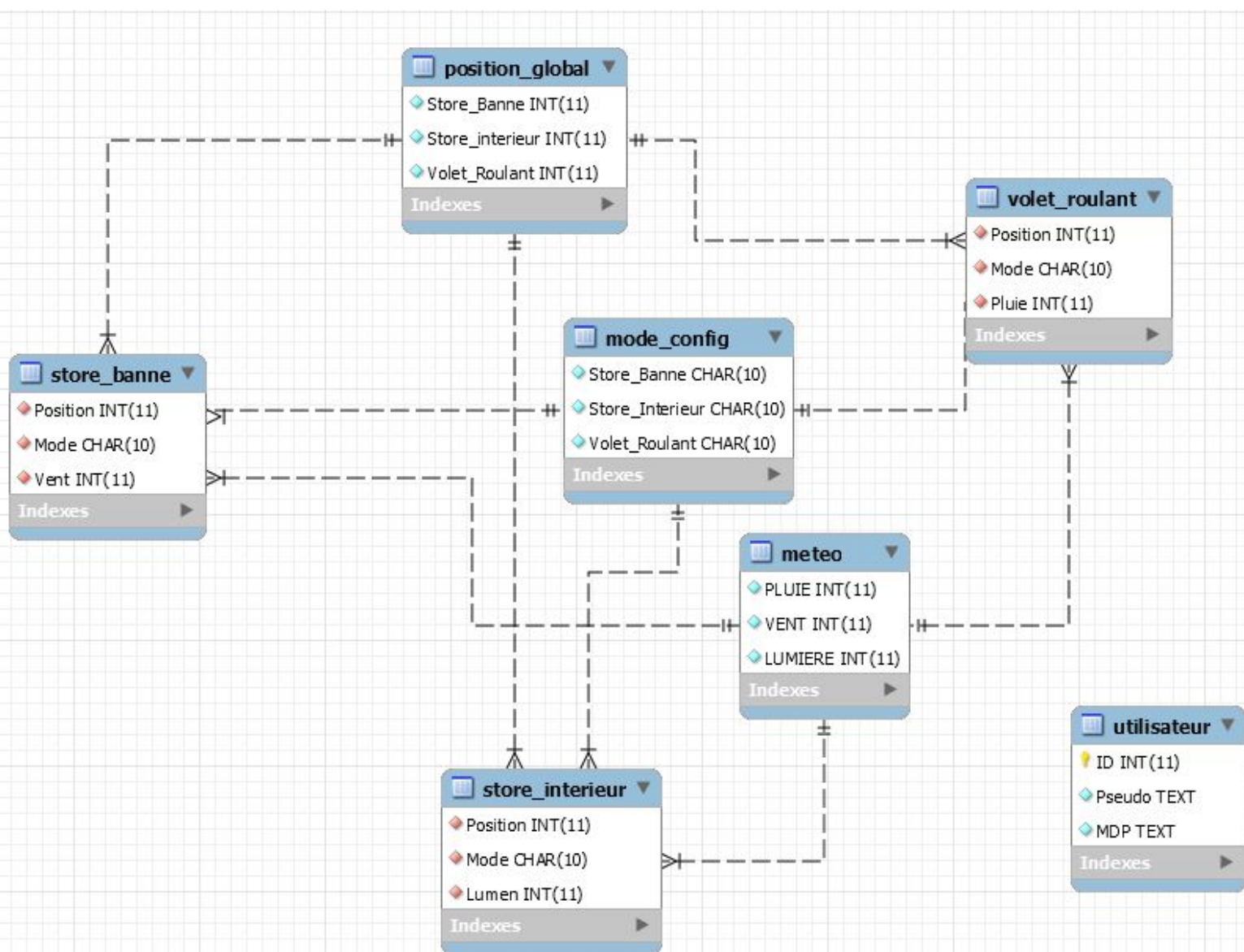
8.5.2.1 QTCP SOCKET :

Cette classe permet, comme indiqué ci dessus, de créer une socket. Une socket possède le même principe de fonctionnement qu'une discussion humaine. (Surtout en mode connecté). Le serveur se met d'abord en mode passif (c'est a dire qu'il attend la connexion d'un client), puis lorsqu'un client tente de se connecter, si les conditions sont exactes, le serveur accepte le client (avec la fonction `accept()`), puis la communication entre le client et le serveur s'établit, permettant de s'échanger des informations grâce au fonctions `send()` et `recv()`.

8.6 Base de Données:

La Base de données a pour rôle de stocker les informations envoyées par les différents capteurs, ainsi que de stocker les exigences de l'utilisateur

La base de données que j'ai créée, représenté ci dessous, stocke les différentes informations que notre applications a besoin, tel que la position ou bien les données des capteurs (stockées dans la table météo).



Les tables volet_roulant, store_banne ainsi que store_interieur, possèdent chacune les mêmes attributs, qui sont Position, Mode et les données du capteur qui leur est lié.

J'ai donc décidé de créer 3 tables, ici position_global, météo et mode_config, dans lesquelles sont stockées toutes les informations.

Ensuite chacune des données est renvoyées dans les tables store_banne, volet_roulant, etc.

Cela permet de ne faire un update que sur une seule table pour changer le mode, et l'utilisateur ne fera qu'un select sur les tables équipements afin de voir la valeur pour chaque équipements.

8.6.1 Communication BDD:

La communication avec la base de données se fait avec la classe QSqlDatabase, qui est fourni par QT.

La première étape est de configurer le driver utiliser pour la communication, ici, étant donné que nous utilisons une base de données stockées sous MySQL, Nous utiliserons le driver Qmysql. D'autre driver sont disponible, tel que QODBC, un intergiciel qui permet a une application de pouvoir se connecter a plusieurs base de données mise en place par des SGBD (tels que MySQL). Pourquoi ne pas utiliser QODBC ? Car ODBC est un système de base de données Windows, et lors de son utilisation, il est relié a un connecteur « MySQL-ODBC », qui va convertir en MySQL.

8.7 Le Site Web:

Le rôle du site web est de permettre au propriétaire, lorsqu'il n'est pas à son domicile, de pouvoir gérer les différents équipements que possèdent son logement.

L'utilisateur distant, celui qui utilise la Base de données, possède les mêmes droits que l'utilisateur local, il peut donc voir depuis son travail ou bien depuis un tout autre endroit le bon fonctionnement de tous les équipements de son logement, et peut aussi interagir avec eux. Il peut aussi depuis le site web vérifier la météo, afin de pouvoir régler les équipements en conséquence.

8.7.1 Langages de Programmation:

8.7.1.1 HTML :

HTML, est le format de données conçu pour représenter les pages web. C'est un langage de balisage permettant d'écrire de l'hypertexte, d'où son nom. HTML permet également de structurer sémantiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie, et des programmes informatiques. Son principal défaut est qu'il ne permet de créer seulement des pages web statiques, sans aucune interaction, alors qu'aujourd'hui plus de 90 % des pages web sont dynamiques. Il est souvent couplé avec PHP pour rendre les pages dynamiques, CSS pour l'aspect graphique est ainsi que du JavaScript pour plus de fluidité.

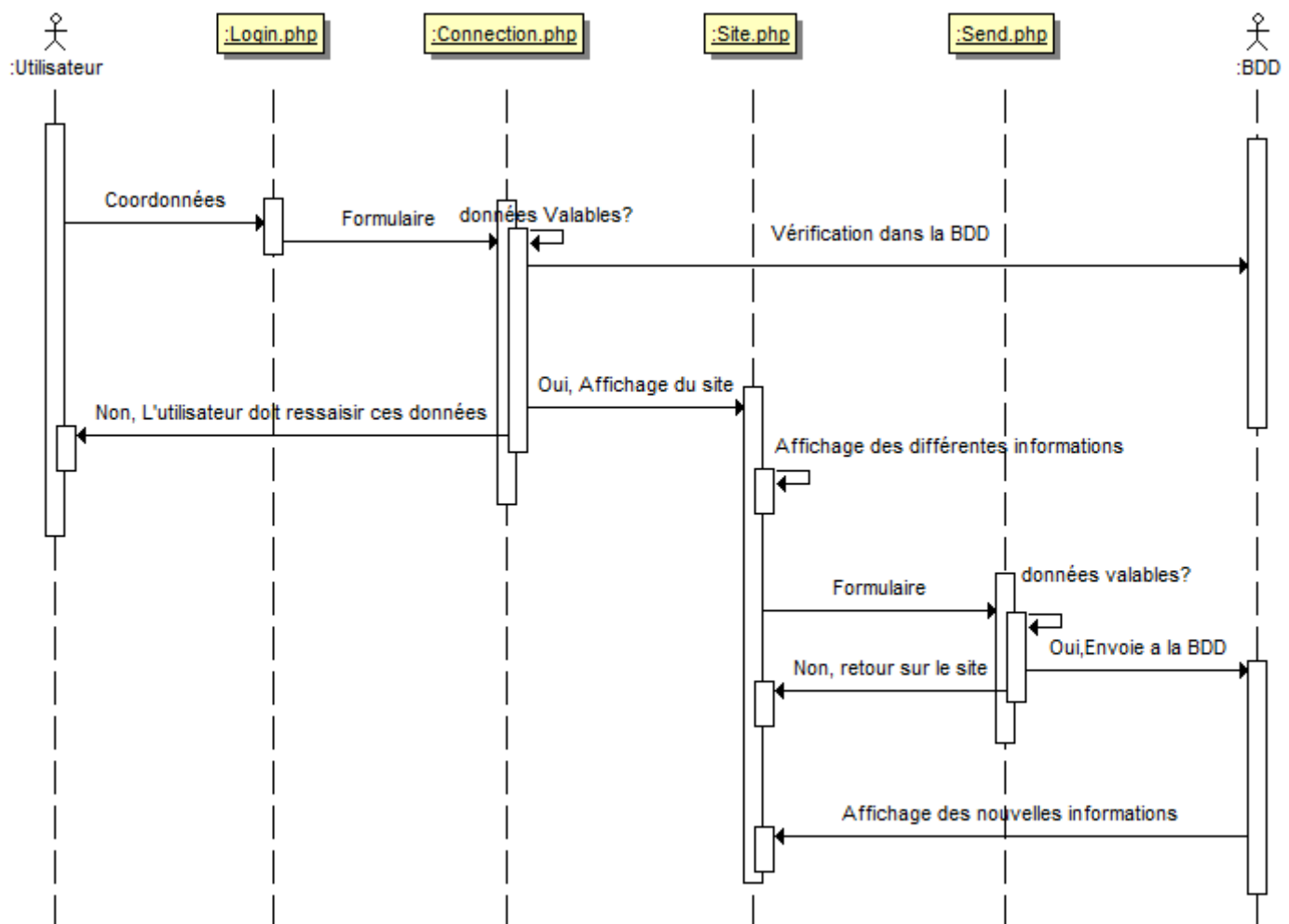
8.7.1.2 PHP :

PHP est un langage de programmation libre, principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. PHP est un langage impératif orienté objet. C'est grâce à ce langage que nous pourrions communiquer avec la base de données dans ce projet.

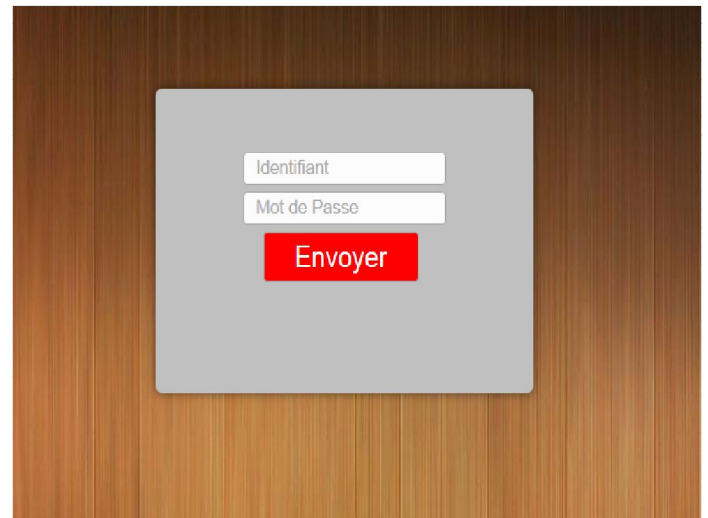
8.7.1.3 CSS :

CSS est un langage de programmation permettant d'embellir son site web, il est souvent couplé avec html, qui possède une interface archaïque. Il permet d'utiliser les feuille de style, qui permettent de modifier la police d'écriture, la couleur, l'arrière plan, ainsi que de rajouter des effets de style sur les menus, etc.

8.7.2 Principe de Fonctionnement:



Comme présenté sur le diagramme de séquence, l'utilisateur doit en premier lieu rentré ces identifiants de connexion. Si les identifiants sont correct, il y a appel de la page Connection.php. dans le cas contraire, l'utilisateur retourne sur la page de login. Le fichier Connection.php permet de traiter les informations qu'à rentrer l'utilisateur, afin de les traiter. Ce fichier permet de savoir si les identifiants de connexions sont exactes, en faisant des vérifications avec la base de données. Comme dit précédemment, si les identifiants sont valides, l'utilisateur est redirigé vers la page principal du site, Site.php.



A login form with a light gray background and rounded corners, centered on a dark wood-textured background. It contains two input fields: 'Identifiant' and 'Mot de Passe', both with light gray borders. Below the fields is a red button with the text 'Envoyer' in white.

Sur cette page, l'utilisateur peut voir les différentes données de chacun des équipements, que ce soit leur position, leur mode ou bien la valeur des différents capteurs (ici des valeurs aléatoire).

Les données sont affichés a chaque fois que la page est appelé ou rafraichi, ce qui permet qu'a chaque fois que l'utilisateur envoie des données grâce au formulaire se trouvant a coté, les données s'afficheront « immédiatement », car la page sera rappelé.



A dashboard interface with a dark wood-textured background. It features three tables of equipment data and a control form on the right.

Position	Mode	Lumen
14	AUTO	96

Position	Mode	Vent
25	MANUEL	78

Position	Mode	Pluie
78	AUTO	24

Position

Mode

Store_Interieur ▼

envoyer

8.7.2.1 Les Formulaires:

Un formulaire est une partie du document constituée d'un contenu dit normal (texte par exemple), ainsi que d'éléments spéciaux appelés commandes (case à cocher, à remplir, bouton de validation).

Par exemple dans ce projet, la page de login est constituée d'un formulaire qui possède deux champs de texte ainsi qu'un bouton de validation.

Voici le code de mon formulaire d'identification.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <link rel="stylesheet" type="text/css" href="index.css" />
  <title>MARI-STOVO</title>
</head>
<form action="connection.php" method="post">
  <p>
    <input type="text" placeholder="Identifiant" name="pseudo"/><br />
    <input type="password" placeholder="Mot de Passe" name="message"/><br />
    <input type="submit" value="Envoyer" name="send" />
  </p>
</form>
</html>
```

Lorsque l'utilisateur appuiera sur le bouton envoyer, les informations qu'il a écrit dans les champs approprié « ici sont identifiant et Mot de passe », les données seront transféré au fichier connection.php, qui vérifiera si les données sont adéquates et permettra donc de continuer sur le site.

8.7.2.2 Envoie de données à la BDD :

L'envoi de données se fait pendant les fichiers PHP intermédiaire, lorsque l'utilisateur rentre ces identifiants ou bien lorsqu'il choisit une configuration pour les différents équipements.

Voici pour représenter mes dires le fichier index.php, celui permettant de vérifier les identifiants.

```
$pseudo =strip_tags($_POST['pseudo']);
$mdp = strip_tags($_POST['message']);

try
{
    $bdd = new PDO('mysql:host=localhost;dbname=MARI_STOVO;charset=utf8', 'root', '');
}
catch(Exception $e)
{
    die('Erreur : '.$e->getMessage());
}

$requete = $bdd->query('SELECT ID FROM utilisateur
    WHERE Pseudo = "'.$pseudo.'" AND MDP = "'.$mdp.'"'');
$donnees=$requete->fetch();

if($donnees['ID']==0)
{
    header('Location:index.php');
}
else
{
    sleep(2);
    header('Location:Site.php');
}
```

Les variables \$pseudo et \$mdp prennent la valeur des variables du formulaire, respectivement pseudo et message.

La variable appelée \$bdd représente la base de données, ainsi que sa configuration. Etant donné que la base de données se trouve sur la carte, ainsi que le site, l'adresse de la base de données est localhost, si la base de données était distante, c'est-à-dire sur un autre PC ou périphérique réseau, il faudrait alors mettre son adresse IP à la place de localhost.

Bien sur si la connexion n'arrive pas à ce faire, un message d'erreur s'affichera sur l'écran de l'utilisateur.

J'envoie ensuite une requête à la base de données, grâce à la fonction fetch(), avec les identifiants reçus de mon formulaire. Si les données sont inexactes, dans le cas présent Pseudo et MDP, l'ID sera égale à zéro, car dans ma base de données ID sert de clé primaire à la table utilisateur.

Si ID n'est pas égale à zéro, ce qui veut dire qu'il y a une correspondance avec la base de données, l'utilisateur sera redirigé vers la page Site.php.

Ce Principe de fonctionnement est aussi appliqué pour ma page d'envoi de donnée, lorsque l'utilisateur envoie des données depuis le formulaire de Site.php.

8.8 Les Capteurs :

8.8.1 Capteur de luminosité :

Le capteur de luminosité possède deux photodiodes, une étant sensible à la lumière visible ainsi que la lumière infrarouge. La deuxième photodiode n'est sensible qu'à la lumière infrarouge. Ces deux valeurs passent ensuite dans un Convertisseur LTD (Light To Digital Converter). afin d'être ensuite utilisées.

Formule du convertisseur : CH0/CH1 (capteurs).

Selon le résultat, la formule pour les lux est modifiée.

Par exemple, si CH0/CH1 est compris entre 0 et 0.52,

La formule pour les lux sera :

$$0,0315 \cdot CH0 - 0,0593 \cdot CH0 \cdot ((CH1/CH0) \cdot 10^{1,4})$$

La valeur retournée, en lux, devra ensuite être échelonnée pour être comprise par l'utilisateur :

Par exemple, 0.5 lux représente une nuit de pleine lune, et 200 lux représente un appartement

Bien éclairé.

L'utilisateur pourra donc sélectionner lui-même les seuils selon ces envies.

8.8.2 Capteur de pluie :

Système de détection Capacitive, cela signifie que lorsque la pluie s'accumule sur le capteur, cela crée une résistance qui empêche toute circulation du courant, signifiant que la pluie est entrain de tomber. Possède une mini résistance chauffante permettant d'éviter les détections intempestives comme le brouillard ou la rosée.

8.8.3 Capteur de Vent :

Anémomètre à coupelle, la vitesse de rotation des coupelles est proportionnelle à la vitesse du vent.

$$V = 2\pi \cdot F(N) \cdot R \cdot N$$

$F(N)$: fonction étalonnage

N : nombre de tour seconde

R : rayon moyen des bras.

La vitesse du vent est en mètre/seconde, il faut multiplier la valeur par 3.6 pour l'avoir en km/heure.