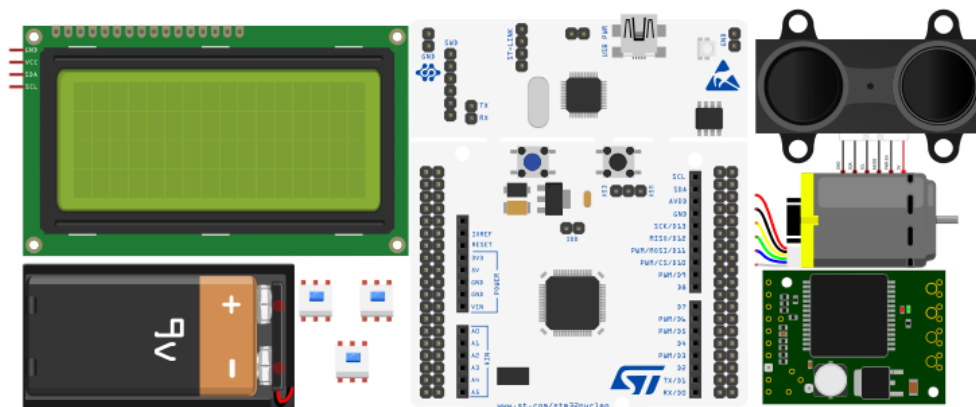


# Projet Lidar



## Compte rendu de projet

Albouy Yann

Version	Date
1.0	02/12/2020
1.1	26/12/2020
1.2	08/01/2021

## **Sommaire :**

<b>1 : Sujet du projet :</b>	<b>3</b>
1.2 : Amélioration du cahier des charges techniques :	3
1.2.1 : Ecran Clear Display 420 :	3
1.2.2 : Moteur à courant continu avec encodeur :	3
<b>2 : Matériel utilisé :</b>	<b>4</b>
2.1 : Carte STM32F411RE :	5
2.1.1 : Ports utilisés :	5
2.2 : Capteur Lidar Lite V3 :	6
2.3 : Bouton poussoir :	6
2.3.1 : Utilisation bouton pour l'interface principale :	6
2.3.2 : Utilisation bouton pour l'interface d'affichage des infos :	7
2.3.2 : Utilisation bouton sur l'interface de saisie de la distance :	7
2.3.3 : Utilisation bouton sur l'interface du mode auto :	8
2.3.4 : Utilisation bouton sur les interfaces de transitions :	8
2.3.5 : Branchement du bouton :	9
2.4 : Ecran LCD Clear Display CLCD 420 :	9
2.4.1 : Commande de l'écran LCD :	9
2.4.2 : Utilisation des sous pixels :	10
2.5 : Contrôleur moteur Pololu md15a :	11
2.6 : Moteur à courant continu avec encodeur :	11
2.6.1 : Connexion à la carte :	12
2.6.2 : Encodeur quadrature 48 CPR:	12
2.6.2.1 : Principe de fonctionnement de l'encodeur en quadrature :	12
<b>3 : Programme :</b>	<b>13</b>
3.1 : Librairie du capteur Lidar :	13
3.2 : Librairie pour les boutons :	13
3.2.1 : Exemple bouton validation :	14
3.2.2 : Utilisation des boutons :	14
3.3 : Librairie pour l'écran LCD :	15
3.3.1 : Interface lcd_navig :	16
3.3.2 : Interfaces au lancement du programme :	17
3.3.3 : Commandes pour l'écran lcd :	18
3.4 : Librairie pour les fonctions globales :	19
3.4.1 : Fonction adaptVitesseDistance :	19
3.4.1.1 : Exemple de la fonction de l'adaptation de vitesse :	20
3.4.2 : Fonction pour le mode auto :	21
3.4.3 : Fonction pour le calcul des tours/min :	23
3.4.3.1 : Graphique variation des RPM :	24
3.5 : Main du programme :	25
<b>4 : Tests unitaires et d'intégration :</b>	<b>26</b>
4.1 : Test pour le capteur lidar :	26

4.2 : Test pour l'écran LCD :	27
4.3 : Test pour les boutons :	27
4.3.1 : Scénario pour l'interface principale :	27
4.3.2 : Scénario pour l'interface d'affichage de la distance et des RPM :	27
4.3.3 : Scénario pour l'interface de saisie de la distance :	28
4.3.4 : Scénario pour l'interface du mode auto :	28
4.3.5 : Scénario pour les interfaces de transitions :	28
<b>5 : Conclusion :</b>	<b>28</b>

# **1 : Sujet du projet :**

L'idée du projet est de programmer une application capable de récupérer à intervalle régulier les données d'un capteur Lidar.

Ensuite il faudra programmer un asservissement de vitesse d'un moteur à courant continu (CC) en fonction de la mesure. Grâce à la distance mesurée par le capteur Lidar, la vitesse s'adaptera. La finalité de ce projet est ainsi de permettre d'adapter la vitesse d'un moteur à courant continu en fonction de la distance que nous retournera le capteur Lidar. Le programme fonctionnera sur une carte sans OS ( Bare-metal ) STM32F411RE Nucleo. La carte n'ayant pas d'OS, il faudra donc partager le temps processeur entre les différents périphériques.

Le système devra aussi pouvoir être autonome et donc avoir une consommation d'énergie raisonnable. On devra aussi avoir un affichage pour vérifier que l'on récupère les bonnes données et vérifier que nos tests sont bons.

## **1.2 : Amélioration du cahier des charges techniques :**

Dans le cahier des charges techniques original, le projet ne contient pas d'écran LCD ni 3 boutons, de même pour le moteur à courant continu avec un encodeur. Ainsi l'écran LCD entre dans le dispositif de contrôle et remplace les leds qui devait permettre de connaître le mode dans lequel on était. De plus, il y a un bouton de plus afin de naviguer dans les différentes interfaces et ainsi permettre plus d'interaction avec l'utilisateur. Le moteur à courant continu simple a aussi été remplacé par un moteur à courant continu avec un encodeur, celui-ci permet de récupérer la vitesse de rotation du moteur.

### **1.2.1 : Ecran Clear Display 420 :**

L'écran clear display CLCD 420 est un afficheur de 4 lignes de 20 caractères rétro-éclairé en bleu. L'écran est livré avec une platine de pilotage permettant la connexion en I2C pour faciliter sa programmation. De plus, grâce à la platine préprogrammée, des commandes sont déjà disponibles.

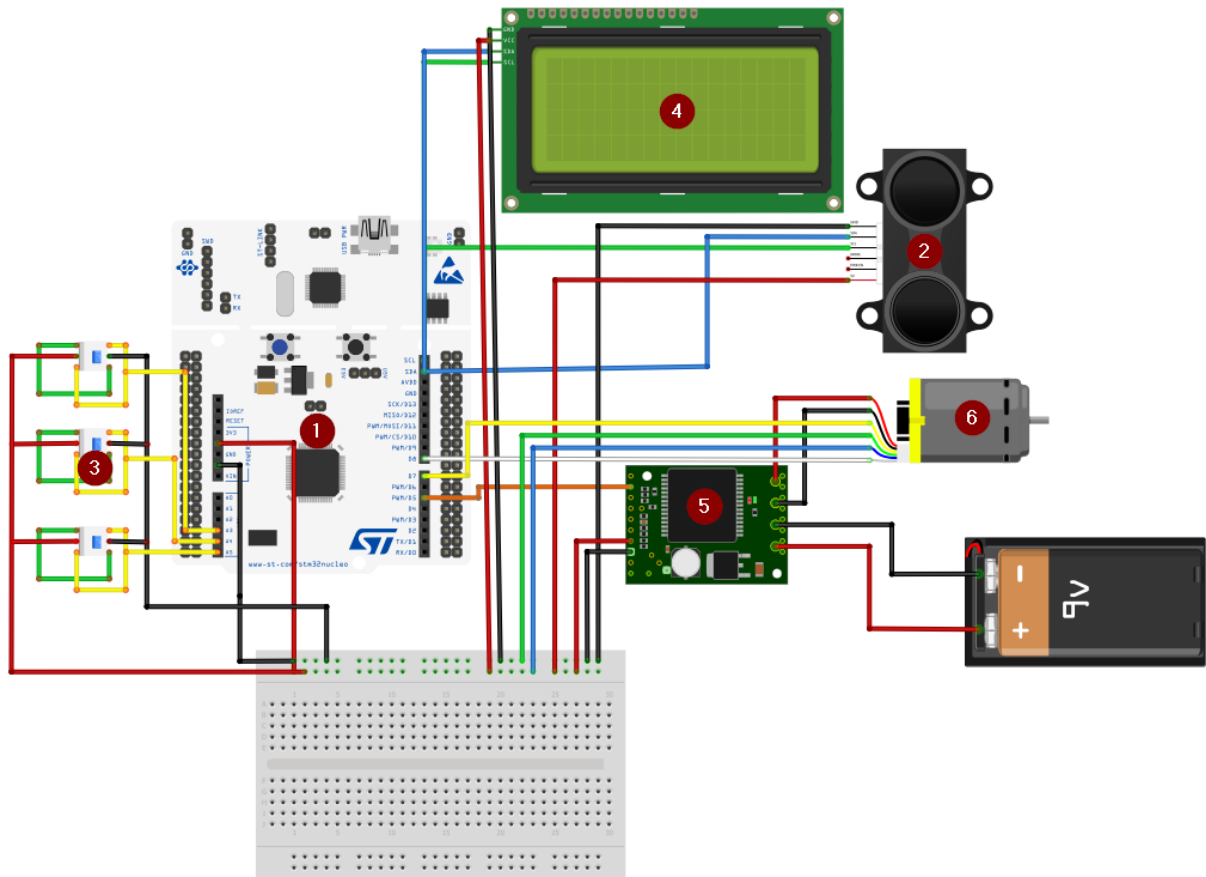
Nom	Tension d'alimentation min	Tension d'alimentation normale	Tension d'alimentation max
CLCD 11020	4.8V	5.0V	5.2V

### **1.2.2 : Moteur à courant continu avec encodeur :**

Ce moteur se compose d'un moteur à courant continu de 6V et d'un réducteur à engrenages droit en métal de 172:1. Il est doté aussi d'un encodeur en quadrature 48 CPR intégré à l'arrière du moteur.

Voltage	Perrformance sans charge	Couple du moteur
6V	34RPM, 250mA	14.5kg.cm, 2.4A

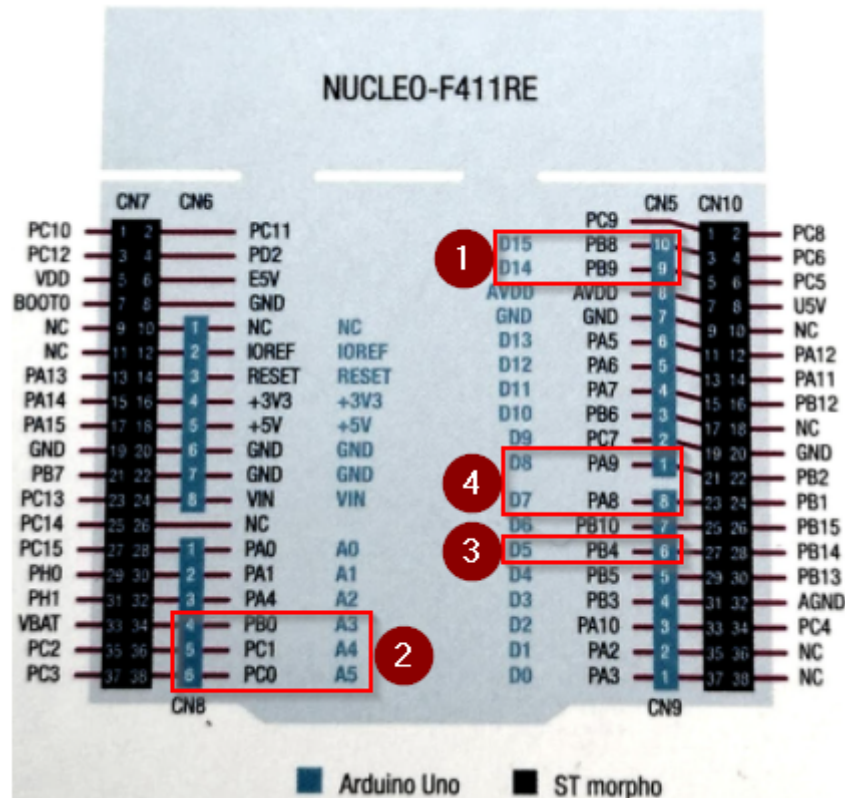
## 2 : Matériel utilisé :



1	<a href="#">Carte STM32F411RE</a>
2	<a href="#">Capteur Lidar Lite V3</a>
3	<a href="#">Bouton Poussoir</a>
4	<a href="#">Ecran Clear Display CLCD 420</a>
5	<a href="#">Contrôleur moteur Pololu md15a</a>
6	<a href="#">Moteur à courant continu avec encodeur</a>

## 2.1 : Carte STM32F411RE :

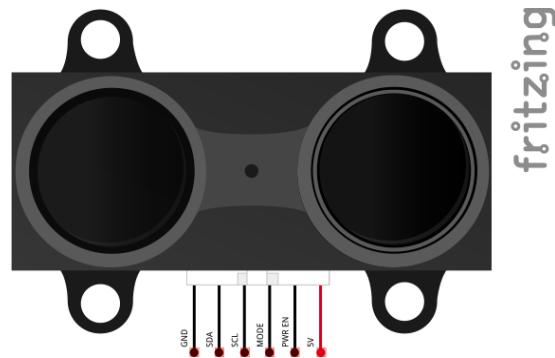
Pour ce projet la carte utilisée est une STM32F411RE. Elle dispose de 512kb de mémoire flash pour le programme et à 128kb de mémoire Ram. Le processeur quant à lui est un Cortex-M4.



### 2.1.1 : Ports utilisés :

<b>1</b> → D14 et D15 / SDA et SCL	Utilisés pour envoyer des commandes au capteur Lidar et récupérer ses données.
<b>1</b> → D14 et D15 / SDA et SCL	Utilisés pour envoyer des commandes à l'écran LCD.
<b>2</b> → Bouton 1	Utilise l'entrée GPIO A5
<b>2</b> → Bouton 2	Utilise l'entrée GPIO A4
<b>2</b> → Bouton 3	Utilise l'entrée GPIO A3
<b>3</b> → Contrôleur Moteur	Envoie de la PWM sur le port D5
<b>4</b> → Encoder Moteur	Récupération des informations de l'encodeur du moteur sur le port D7 et D8

## 2.2 : Capteur Lidar Lite V3 :



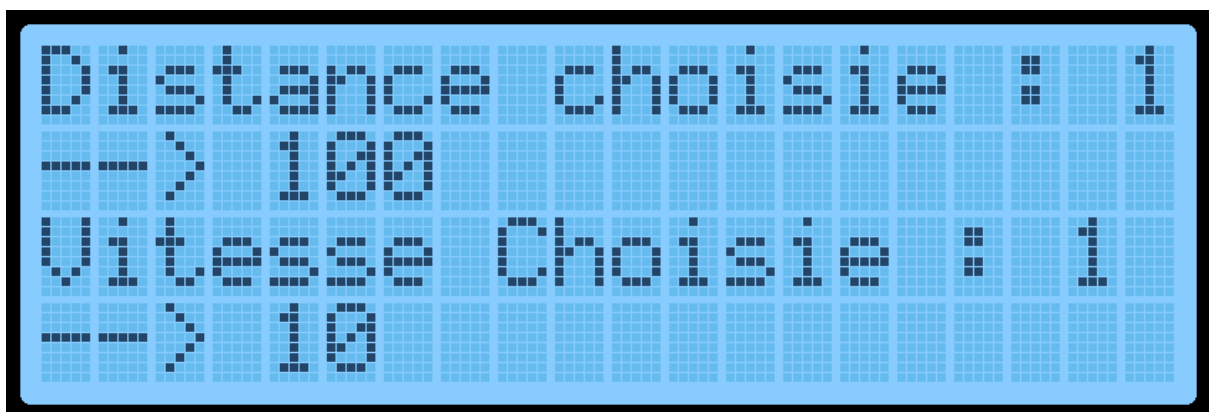
Le capteur Lidar est branché en I2C sur la carte STM32 et il a pour adresse 0x62 à laquelle il faut appliquer un décalage à gauche. Pour calculer une distance, le capteur Lidar émet des impulsions infrarouges et mesure le temps que met l'onde pour revenir. Il permet de calculer précisément une distance. Le capteur utilisé dans le projet est un Lidar de chez Garmin, le Lidar Lite V3. A moins de 5M sa précision est d'environ 2.5cm et au delà de 5m la précision est d'environ une dizaine de centimètres.

## 2.3 : Bouton poussoir :



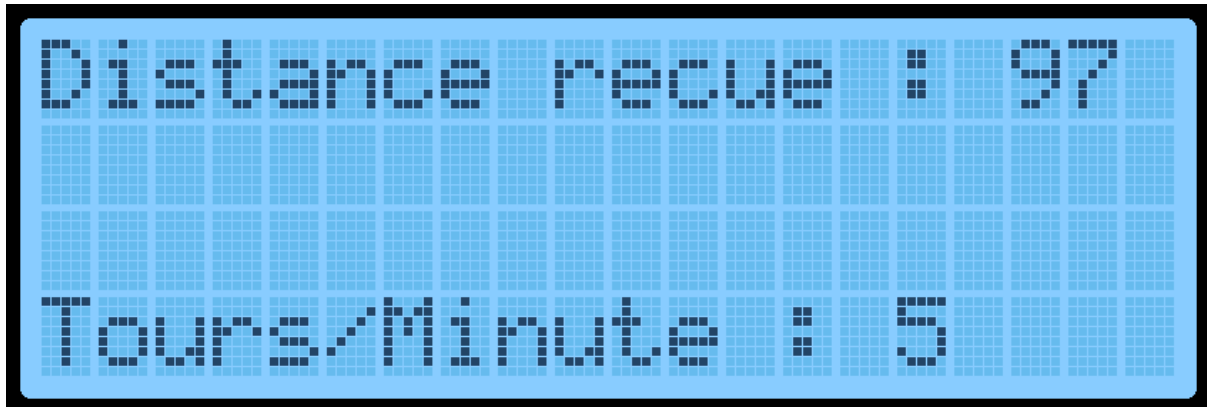
Afin de faciliter l'utilisation du programme et d'avoir une interaction avec l'utilisateur, le projet comporte 3 boutons SPPH410200 de chez Alps Alpine qui en fonction de l'interface auront différentes fonctions. Le premier bouton est branché en GPIO IN sur le port PC0, le second bouton est branché en GPIO IN sur le port PC1 et le troisième bouton est lui branché en GPIO IN sur le port PB0.

### 2.3.1 : Utilisation bouton pour l'interface principale :



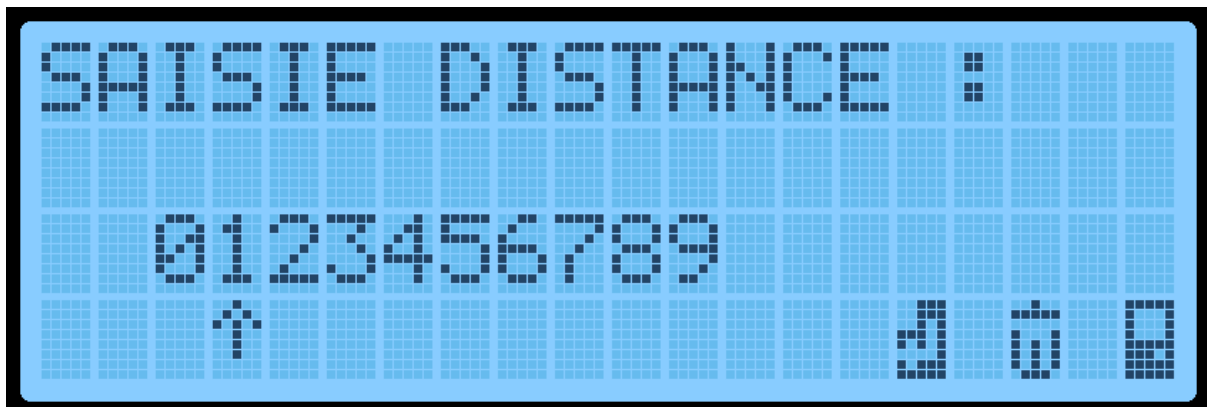
Sur cette interface, le **bouton 1** sert à changer la distance choisie, le **bouton 2** permet de changer la vitesse choisie, et le **bouton 3** lui permet de changer d'interface et d'aller sur l'interface où l'on affiche en temps réel la distance retournée par le capteur lidar et le nombre de rotation du moteur par minute.

### 2.3.2 : Utilisation bouton pour l'interface d'affichage des infos :





Sur l'interface qui affiche en direct les données reçues par le capteur Lidar et l'encodeur du moteur, seul le **bouton 3** est utilisable et permet de retourner sur l'interface principale.

### 2.3.2 : Utilisation bouton sur l'interface de saisie de la distance :



Cette interface utilise les 3 boutons. Le **bouton 1** permet de déplacer le curseur de saisie à gauche, le **bouton 3** permet de déplacer le curseur de saisie vers la droite et enfin le **bouton 2** permet de valider la saisie, ou bien la validation ou la suppression de la distance.

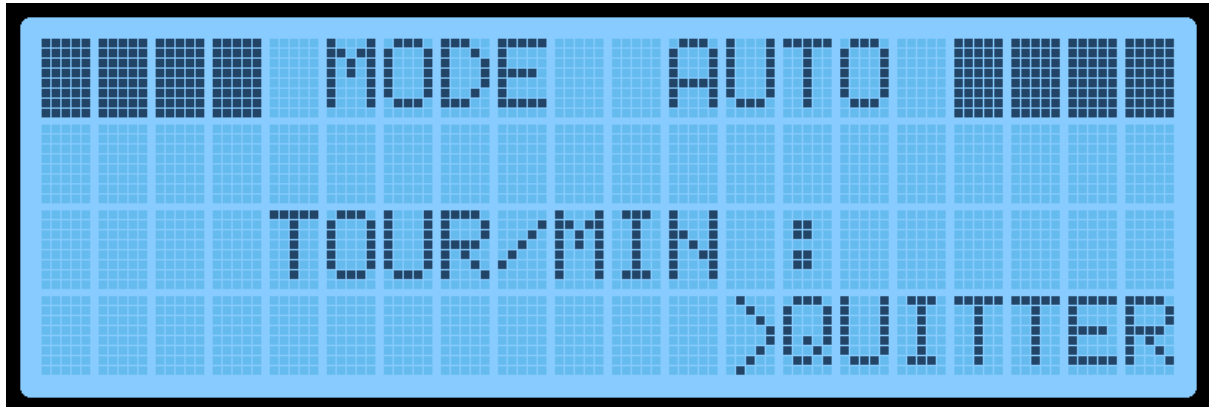
	<p>Cette touche permet de valider le choix de distance fait par l'utilisateur. Quand il valide la distance, il est ramené sur l'interface principale du programme.</p>
	<p>Cette touche permet de supprimer la saisie chiffre par chiffre. Une fois placé sur cette icône, l'utilisateur n'a qu'à valider pour supprimer le ou les nombres qu'il souhaite.</p>





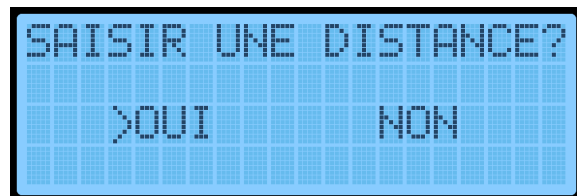
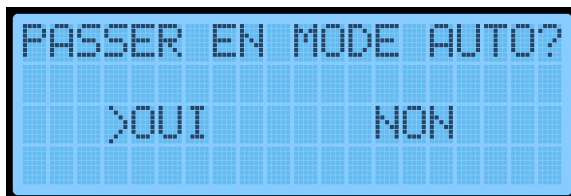
Cette touche permet de sortir de la saisie de distance et de retourner à l'interface principale sans prendre en compte la valeur saisie.

### 2.3.3 : Utilisation bouton sur l'interface du mode auto :



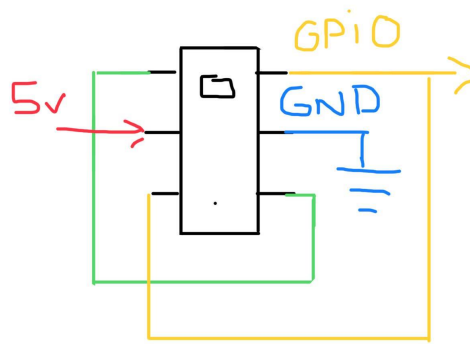
Dans cette interface seul le **bouton 2** servira à quitter le mode auto pour retourner à l'interface principale. Cette interface permet de passer la vitesse en mode automatique en fonction de la distance qu'aura sélectionné l'utilisateur auparavant. Ainsi il pourra voir en direct la vitesse en Tour/min s'ajuster en fonction de la distance.

### 2.3.4 : Utilisation bouton sur les interfaces de transitions :



Pour cette interface on utilisera le **bouton 1** pour sélectionner le choix "OUI" ou "NON" et le **bouton 2** servira à valider le choix. Si l'utilisateur fait le choix de cliquer sur "NON" il sera ramené à l'interface principale du programme, tandis que s'il clique sur "OUI" il sera alors soit amené sur l'interface du mode auto, soit sur l'interface pour la saisie de la distance.

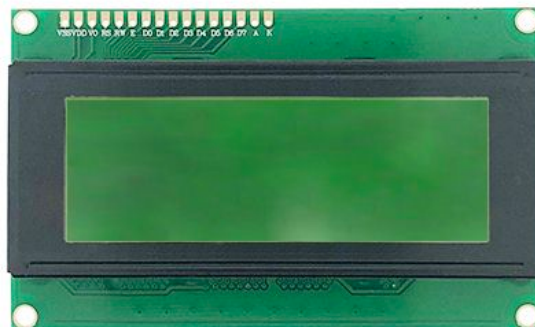
### 2.3.5 : Branchement du bouton :



Ce branchement permet d'avoir un "clic" du bouton, ainsi a chaque pression on pourra exécuter ce que l'on souhaite. Le fil jaune vers la sortie GPIO mene sur les différents ports de la carte STM32.

- **Bouton 1 : GPIO A5**
- **Bouton 2 : GPIO A4**
- **Bouton 3 : GPIO A3**

### 2.4 : Ecran LCD Clear Display CLCD 420 :



L'écran LCD a une taille de 20\*4. Il est branché en I2C sur le port D14 et D15 de la carte STM32. Son adresse est 0x00 avec le décalage à gauche. Par défaut, l'écran a des caractères spéciaux stockés dans sa mémoire et l'utilisateur peut en personnaliser 8, soit de l'adresse 0x08 à 0x0F. Sur cette plage l'utilisateur pourra définir les sous pixels pour afficher ce qu'il souhaite.

#### 2.4.1 : Commande de l'écran LCD :

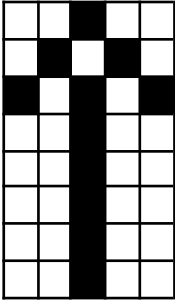
Il est possible d'envoyer à l'écran différentes commandes afin d'interagir avec lui, il suffit de lui envoyer la commande souhaitée au travers d'un `HAL_I2C_Master_Transmit`.

Commande	Effet de la commande
0x1B 0x43	Efface l'écran, laisser un délai de 15 ms
0x1B 0x53	Active le curseur sur l'écran

0x1B 0x73	Désactive / Enlève le curseur de l'écran
0x1B 0x42	Active le rétro-éclairage
0x1B 0x62	Désactive le rétro-éclairage
0x1B 0x48	Place le curseur en haut à gauche en position 0,0
0x1B 0x4C 0xXX 0xYY	Placer le curseur à la position X et Y
0x01	Se placer à la première ligne
0x02	Se placer à la seconde ligne
0x03	Se placer à la troisième ligne
0x04	Se placer à la quatrième ligne

#### 2.4.2 : Utilisation des sous pixels :

En plus des caractères spéciaux stockés par défaut dans la mémoire de l'écran, l'utilisateur peut aussi en créer. Pour cela il doit suivre la démarche suivante :  
Définir ce que l'on veut afficher et qui fait une taille de 5\*8.



Ensuite pour chaque ligne on vas faire la conversion en Hexa :

**Exemple pour la troisième ligne :**

16	8	4	2	1
1	0	1	0	1

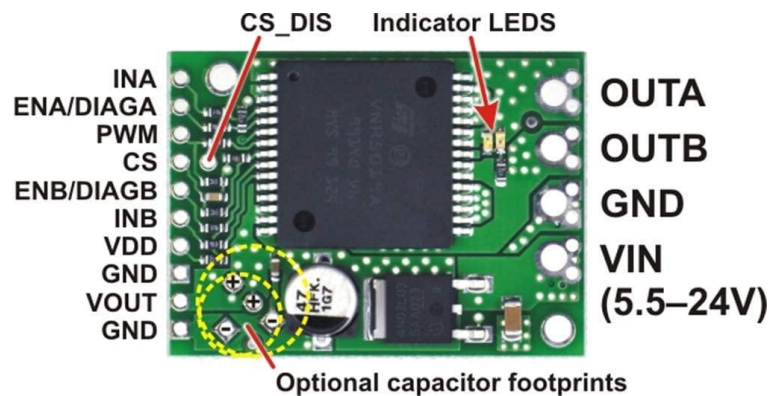
La valeur à saisir pour la commande de la troisième ligne sera : 0x15

Il suffira de répéter cela pour les 8 lignes, pour la flèche ci-dessus on obtiendra donc :

Commande	Adresse ou on enregistre le pixel	Code en Hexa obtenu à partir de notre dessin
0x1B 0x44	0x08	0x04 0x0A 0x15 0x04 0x04 0x04 0x04 0x04

Notre pixel sera enregistré à l'adresse 0x08 et il suffira de l'appeler pour l'afficher sur l'écran.

## 2.5 : Contrôleur moteur Pololu md15a :



Ce composant permet d'envoyer une PWM à notre moteur pour contrôler sa vitesse de rotation, grâce à la PWM qu'on lui envoie la vitesse de rotation varie.

OUTA	Connecter à un pin du moteur
OUTB	Connecter à un pin du moteur
GND	Brancher sur le - d'une alimentation externe (pile 9v).
VIN	Brancher sur le + de l'alimentation externe
INA	Brancher sur le +3.3V, pour alimenter le contrôleur.
PWM	Brancher sur le port D5 qui envoie la PWM

## 2.6 : Moteur à courant continu avec encodeur :



[www.pololu.com](http://www.pololu.com)

Ce moteur comporte un encodeur qui permet de récupérer le nombre de rotation du moteur, et un réducteur qui a un rapport de 172:1, ce qui signifie que l'axe en sortie du moteur doit faire 172 tours pour que l'axe à la sortie du réducteur en fasse 1.

### 2.6.1 : Connexion à la carte :

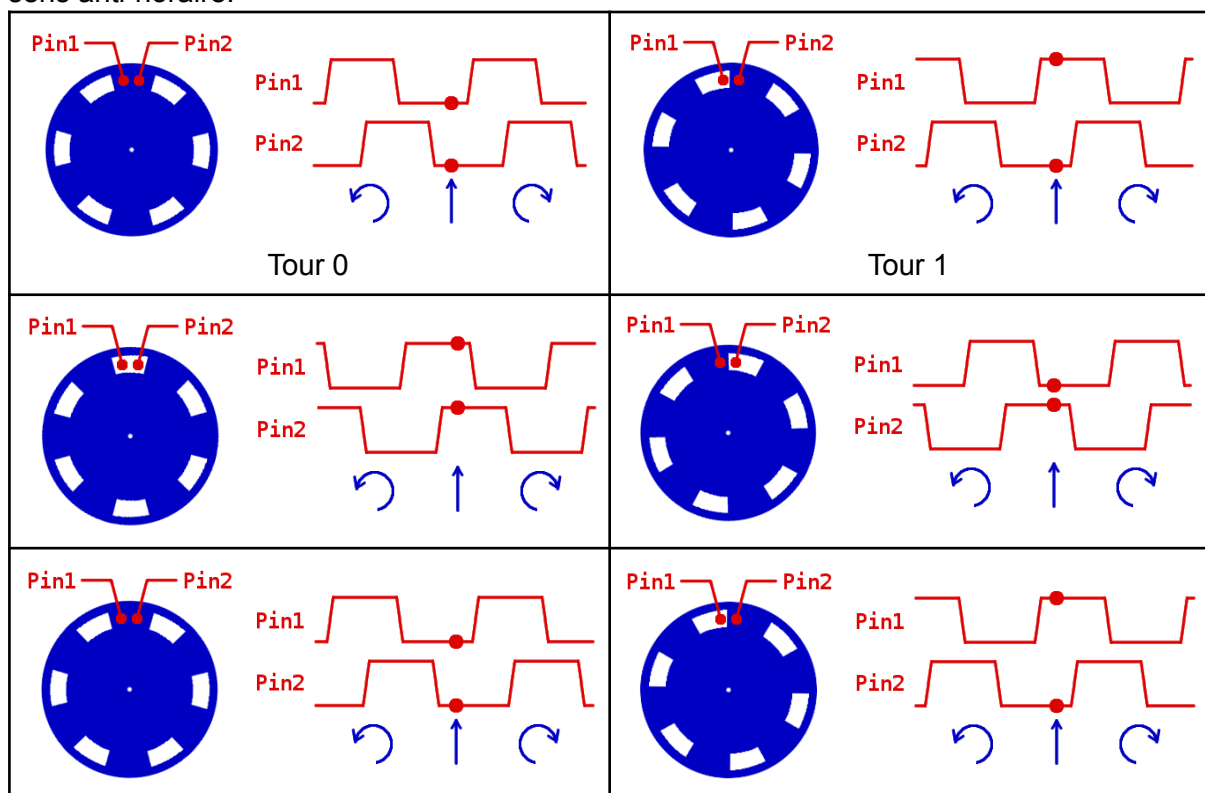
Rouge	+5v à connecter au moteur
Noir	Ground du moteur
Vert	Ground de l'encodeur du moteur
Bleu	+5v de l'encodeur du moteur
Jaune	Sortie <b>A</b> de l'encodeur branché sur le port <b>D7</b>
Blanc	Sortie <b>B</b> de l'encodeur branché sur le port <b>D8</b>

### 2.6.2 : Encodeur quadrature 48 CPR:

Un encodeur en quadrature à 48 CPR (Count Per Revolution ) signifie qu'à chaque fois que l'axe du moteur a fait un tour on a 48 compte par révolution. Dans le cas de notre moteur qui a un réducteur de 172:1, on aura ainsi  $172 \times 48 = 8256$  comptage pour que l'axe en sortie du réducteur fasse un tour complet.

#### 2.6.2.1 : Principe de fonctionnement de l'encodeur en quadrature :

Dans le cas suivant, le moteur tourne dans le sens horaire, mais cela marche aussi dans le sens anti-horaire.



L'encodeur peut détecter les mouvements du moteur dans les 2 sens grâce aux marques qui passent dans les différentes positions du moteur. Dans le sens horaire, quand le disque (bleu) de l'encodeur tourne, les changements sont d'abord détectés par le pin 1, puis le pin

2. L'encodeur est dit en quadrature car les signaux entre les 2 pins sont déphasés de 90 degrés.

## **3 : Programme :**

### **3.1 : Librairie du capteur Lidar :**

Dans la librairie de mon capteur lidar on y retrouve une fonction pour l'initialiser, celle-ci permet de mettre en route et de configurer le capteur Lidar. Pour faire cela, il suffit d'envoyer des commandes en I2C au capteur lidar.

```
void lidar_init(void)
{
    /*-----INIT DU CAPTEUR LIDAR-----*/
    uint8_t cmd[1];
    cmd[0]=0x04;
    HAL_I2C_Mem_Write(&hi2c1, LIDAR_ADD ,0x00, 1, cmd,1,100);

    /*-----CONFIGURATION LIDAR-----*/
    cmd[0]=0xff;
    HAL_I2C_Mem_Write(&hi2c1, LIDAR_ADD,0x02, 1, cmd,1,1000);
    cmd[0]=0x08;
    HAL_I2C_Mem_Write(&hi2c1, LIDAR_ADD,0x04, 1, cmd,1,1000);
    cmd[0]=0x00;
    HAL_I2C_Mem_Write(&hi2c1, LIDAR_ADD,0x1c, 1, cmd,1,1000);
    /*-----FIN DE CONFIGURATION LIDAR-----*/
}
```

### **3.2 : Librairie pour les boutons :**

Dans cette librairie on retrouve les différentes fonctions permettant de faire fonctionner les boutons en fonction de l'interface choisie par l'utilisateur. Cette librairie permet avec 3 boutons d'avoir plusieurs types d'interactions possibles. On retrouve dans cette interface plusieurs fonctions pour les boutons afin de conserver les choix de l'utilisateur tout au long du programme.

```
/* Fonction pour connaître la distance choisie */
/* Prend en INT un entier */
/* Retourne un entier */
int boutonDistance(int);

/* Fonction pour connaître la vitesse choisie */
/* Prend en INT un entier */
/* Retourne un entier */
int boutonVitesse(int);
```

```

/* Fonction pour connaitre Le choix utilisateur */
int boutonChoix(int);

/* Fonction pour choisir La distance avec Le bouton */
void clicBtnDistance(void);

/* Fonction pour choisir La vitesse avec Le bouton */
void clicBtnVitesse(void);

/* Fonction pour choisir L'interface avec Le bouton */
void clicBtnInterface(void);

/* Fonction pour choisir La reponse avec Le bouton */
void clicBtnChoix(void);

/* Fonction pour valider Le choix fait par L'utilisateur */
void clicBtnValide();

/* Fonction pour faire bouger La fleche a gauche */
void clicBtnGauche(void);

/* Fonction pour faire bouger La fleche a droite */
void clicBtnDroit(void);

```

Le squelette des boutons est toujours le même à quelques différences près, chacun doit avoir une variable à incrémenter en fonction du cas d'utilisation et une variable pour éviter le double clic. Ainsi dans le code de nos boutons les seules différences se trouveront sur les variables appelé **onePush** et **button\*\*\*\*\*** désignant la variable à incrémenter. Pour ce qui est du reste du code du bouton on y retrouve un clear de l'écran actif ou non en fonction du besoin et une sortie de veille en fonction du cas aussi. La variable **écranActif** permet de mettre en veille l'écran au bout de 30 secondes.

### 3.2.1 : Exemple bouton validation :

```

void clicBtnValide(void)
{
    if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_1) == 1 && onePush2 == 0)
    {
        lcd_on();
        lcd_clear();
        ecranActif = HAL_GetTick();
        onePush2++;
        buttonValide = buttonValide +1;
    }
    else if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_1) == 0){
        onePush2 = 0;
    }
}

```

### 3.2.2 : Utilisation des boutons :

Pour utiliser les boutons dans notre programme, il suffit d'appeler la fonction du bouton que l'on souhaite et d'inclure au début du programme la librairie **libBouton.h** comme dans

l'exemple ci-dessous avec l'interface de transition pour passer en mode auto dans le programme.

```
#include "libBouton.h"
/*-----CODE-----*/
void lcd_transition(void)
{

    /*-----BOUTON POUR CHANGER LE CHOIX-----*/
    clicBtnChoix();
    /*-----FIN DE LA PARTIE CONCERNANT LE BOUTON POUSSOIR DES CHOIX-----*/

    /*-----BOUTON POUR VALIDER LE CHOIX-----*/
    clicBtnValide();
    /*-----FIN DE LA PARTIE CONCERNANT LE BOUTON POUSSOIR DE VALIDATION-----*/

    if(buttonChoix == 1)
    {
        if(buttonValide == 1)
        {
            /*-----ACTION A FAIRE-----*/
        }
    }
    else if(buttonChoix == 2)
    {
        if(buttonValide == 1)
        {
            /*-----ACTION A FAIRE-----*/
        }
    }
    else if(buttonChoix == 3)
    {
        /*-----ACTION A FAIRE-----*/
    }
}
```

L'appel des boutons se fait donc au début de la fonction, et on utilisera la variable à incrémenter du bouton "**button\*\*\*\*\***" dans la fonction.

### 3.3 : Librairie pour l'écran LCD :

Grâce à cette librairie, on peut afficher les informations utiles sur l'écran. On retrouve dans celle-ci les différentes interfaces qui seront utilisées par le programme.

```
/* Fonction pour verifier connection LCD*/
int lcd_isConnected(void);

/* Fonction d'init de mon ecran LCD */
void lcd_init (void);
```



```

/* Fonction pour clear le LCD */
void lcd_clear (void);

/* Fonction pour afficher la distance du Lidar */
void lcd_affDistance(int,double);

/* Fonction affichage distance et vitesse */
void lcd_interfacePrinc(int,int,int);

/* Fonction pour allumer l'écran LCD */
void lcd_on(void);

/* Fonction pour éteindre l'écran LCD */
void lcd_off(void);

/* Fonction pour faire une voiture sur le LCD */
void lcd_maVoiture(void);

/* Fonction pour mettre l'écran en veille */
void lcd_Veille(int);

/* Fonction pour faire le mode auto de vitesse */
void lcd_ModeAuto(double);

/* Fonction pour naviguer entre mes interfaces */
void lcd_Navig();

/* Fonction qui fera la transition avec le choix de l'utilisateur pour le mode auto */
void lcd_transition(void);

/* Fonction qui fera la transition avec le choix de l'utilisateur pour la saisie de distance */
void lcd_transitionDistance(void);

/* Fonction pour saisir une distance personnalisée entre 1 et 40M */
void lcd_SaisieDistance(void);

/* Fonction affichage valeur trop grande ou trop petite */
void lcd_mauvaiseSaisie(int);

```

### 3.3.1 : Interface lcd\_navig :

L'interface **lcd\_Navig** est le "sommaire" des interfaces de notre programme, c'est grâce à cette fonction que l'on pourra naviguer dans les différentes interfaces en fonction des choix de l'utilisateur. On peut voir dans le code de la fonction qu'on utilise ici le bouton "**clicBtnInterface()**" qui va nous permettre d'accéder en fonction du nombre de clic aux différentes interfaces. On remarque aussi qu'il y a des interfaces qui ne sont pas atteignables avec le bouton de changement d'interface. On ne peut pas y avoir accès normalement car ces interfaces ne sont utiles que dans des cas précis et ne doivent pas être accessibles tout le temps. Ce sont les interfaces pour :

- Les transitions pour saisir une distance où aller en mode auto
- L'interface pour le mode auto où la saisie de distance
- L'interface qui informe l'utilisateur en cas de mauvaise saisie pour la distance

```

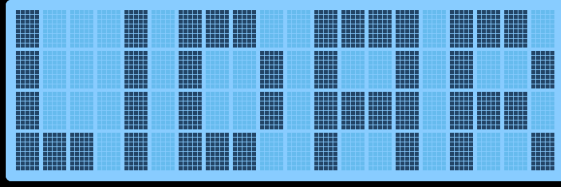
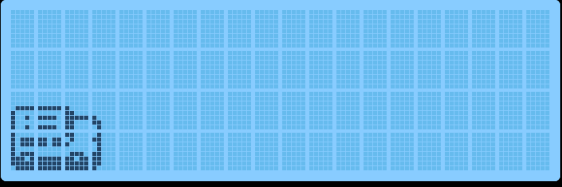
void lcd_Navig()
{
    /*---DESACTIVER CURSEUR MOCHE---*/
    uint8_t cmdCursOFF[2]={0x1B, 0x73};
    HAL_I2C_Master_Transmit(&hi2c1,LCD_ADD,cmdCursOFF,2,100);
    HAL_Delay(1);

    /*---BOUTON POUR CHANGER D'INTERFACE---*/
    clicBtnInterface();
    /*---FIN DE LA PARTIE CONCERNANT LE BOUTON POUSSOIR DE L'INTERFACE---*/
    if(buttonInterface == 1 || nb == 1)
    {
        adaptVitesseDistance(boutonDistance(buttonDistance) , boutonVitesse(buttonVitesse) );
        lcd_interfacePrinc(buttonDistance, boutonVitesse,tpsCourant);
        /*---MISE EN VEILLE---*/
        lcd_Veille(ecranActif);
    }
    else if(buttonInterface == 2 || nb == 2)
    {
        adaptVitesseDistance(boutonDistance(buttonDistance) , boutonVitesse(buttonVitesse) );
        lcd_affDistance(distance, rpm);
        /*---MISE EN VEILLE---*/
        lcd_Veille(ecranActif);
    }
    else if(buttonInterface == 3 || nb == 3)
    {
        boutonInterface = 1;
    }
    /*---INTERFACE NON VISIBLE PAR SELECTION---*/
    else if(buttonInterface == 6 || nb == 6)
    {
        lcd_transition();
    }
    else if(buttonInterface == 7 || nb == 7)
    {
        modeAuto(distance, boutonDistance(buttonDistance));
        lcd_ModeAuto(rpm);
    }
    else if(buttonInterface == 8 || nb == 8)
    {
        lcd_transitionDistance();
    }
    else if(buttonInterface == 9 || nb == 9)
    {
        lcd_SaisieDistance();
    }
    else if(buttonInterface == 10)
    {
        lcd_mauvaiseSaisie(maDist);
    }
}

```

### 3.3.2 : Interfaces au lancement du programme :

Dans la librairie de l'écran Lcd on retrouve aussi les interfaces qui s'afficheront au lancement du programme afin d'avoir une animation au lancement. On aura ainsi la fonction **lcd\_init** et **lcd\_maVoiture**. Elles sont lancées dans le main du programme avant la boucle infinie et après avoir vérifié que l'écran soit connecté.

	
<p>La première interface à apparaître est <b>lcd_init</b> qui affiche sur l'écran le nom du projet, et disparaît ensuite en glissant vers le bas.</p>	<p>Viens ensuite l'interface <b>lcd_maVoiture</b> qui fait apparaître une voiture en bas à gauche de l'écran, et qui va se déplacer vers la droite. Elle fera 2 fois le tour de l'écran avant que l'utilisateur puisse accéder à l'interface principale. Le nombre de tours est configurable dans la fonction <b>lcd_maVoiture</b> à la ligne 1206.</p>

### 3.3.3 : Commandes pour l'écran lcd :

Dans la librairie pour l'écran LCD on retrouve aussi des fonctions qui envoient simplement des commandes afin d'allumer / éteindre l'écran, ou simplement le clear. Il y a aussi une fonction pour avoir une veille afin d'économiser de l'énergie.

```
void lcd_clear(void)
{
    uint8_t cmdLDCLEAR[2]={0x1B, 0x43};
    HAL_I2C_Master_Transmit(&hi2c1,LCD_ADD,cmdLDCLEAR,2,100);
    HAL_Delay(15);
}

void lcd_on(void)
{
    uint8_t cmdON[2]={0x1B, 0x42};
    HAL_I2C_Master_Transmit(&hi2c1,LCD_ADD,cmdON,2,100);
    HAL_Delay(1);
}

void lcd_off(void)
{
    uint8_t cmdOFF[2]={0x1B, 0x62};
    HAL_I2C_Master_Transmit(&hi2c1,LCD_ADD,cmdOFF,2,100);
    HAL_Delay(1);
}
```

```

void lcd_Veille(int tpsActif)
{
    /*----MISE EN VEILLE AU BOUT DE 30S----*/
    /*----    MODIF TEMPS EN MS    ----*/
    if(tpsCourant - tpsActif >= 30000)
    {
        lcd_off();
    }
}

```

Pour ce qui est de la fonction de mise en veille, elle prend en paramètre le temps actif qui correspond au temps qui s'est écoulé depuis le lancement du programme et qui est stocké dans une variable lors de l'appui sur un bouton si celui-ci gère la veille. Le temps étant en ms, on fait ensuite la différence entre la valeur stockée et le temps courant qui lui continue d'évoluer à mesure que le programme tourne. Dans notre cas, l'écran passera en veille au bout de 30 secondes, ou 30000ms.

### 3.4 : Librairie pour les fonctions globales :

Dans cette librairie, on retrouve les fonctions d'asservissement du moteur. Ainsi la première est une fonction qui prend en compte les choix faits par l'utilisateur sur l'interface principale afin d'ajuster la vitesse du moteur, c'est la fonction **adaptVitesseDistance**. On retrouve ensuite la fonction pour le **mode auto**, et celle qui permet de **calculer le nombre de tour/min** du moteur.

#### 3.4.1 : Fonction adaptVitesseDistance :

Cette fonction prend en paramètres deux entiers, le premier correspond à la distance choisie par l'utilisateur à l'aide de l'interface et des boutons, et le second entier lui correspond au pourcentage de la puissance du moteur choisi. A l'intérieur de cette fonction on retrouve 3 conditions, la première est le cas où l'on est à moins de 80 cm d'un obstacle, le moteur s'arrête. Le second cas est quand on a une distance supérieure à la distance choisie, alors on pourra accélérer jusqu'à la vitesse choisie par l'utilisateur. Le dernier cas permet de ralentir lorsqu'on a une distance inférieure à celle choisie par l'utilisateur, le moteur va donc ralentir progressivement jusqu'à l'arrêt complet si on passe en dessous de 80 cm, sinon on maintiendra ou accélérera en fonction de la distance.

```

void adaptVitesseDistance(int choixDistance, int choixVitesse)
{
    double diffDistance = 0;
    double nouvVitesse = 0;
    /*----CAS ARRET DU MOTEUR CAR TROP PROCHE----*/
    if( distance <= 80)
    {
        /*----ARRET----*/
        htim3.Instance->CCR1 = 0;
    }
    /*----MISE A LA VITESSE SOUHAITE SI ON PEUT----*/
    if( distance >= choixDistance)
    {
        htim3.Instance->CCR1 = choixVitesse;
    }
    /*----VARIATION VITESSE POUR RALENTIR SI L'OBSTACLE SE RAPPROCHE----*/
}

```

```

if( distance < choixDistance)
{
    /*---CALCUL POUR REDUIRE LA VITESSE---*/
    diffDistance = choixDistance - distance;
    /*---ATTENTION SI ECART TROP IMPORTANT VAL NEGATIVE---*/

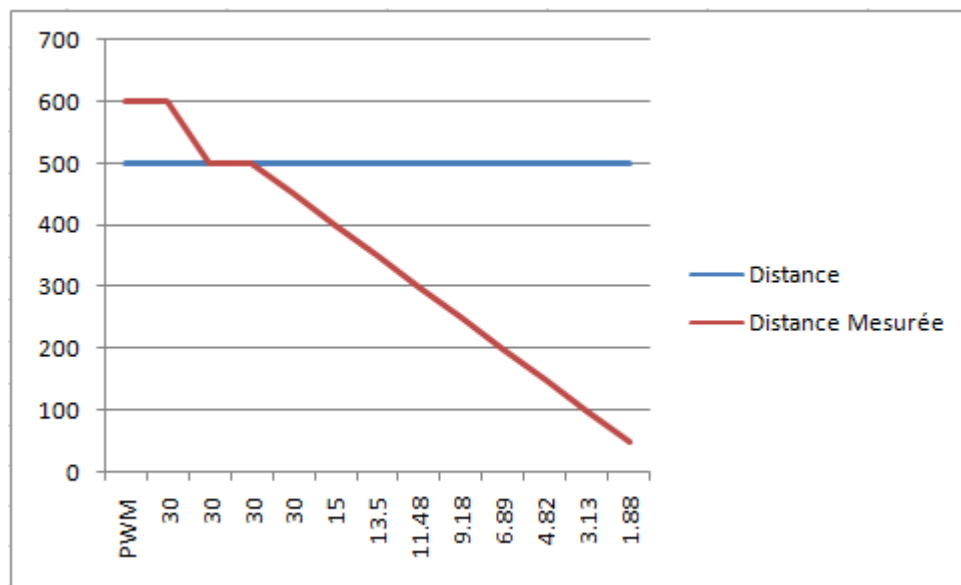
    /*---GESTION EN CAS DE VALEURS NEGATIVES---*/
    if(diffDistance > 100)
    {
        nouvVitesse = 1-(diffDistance/1000);
    }
    else if(diffDistance > 1000)
    {
        nouvVitesse = 1-(diffDistance/10000);
    }
    else{
        nouvVitesse = 1-(diffDistance/100);
    }
    nouvVitesse = (choixVitesse*nouvVitesse);

    htim3.Instance->CCR1 = nouvVitesse;
}
}

```

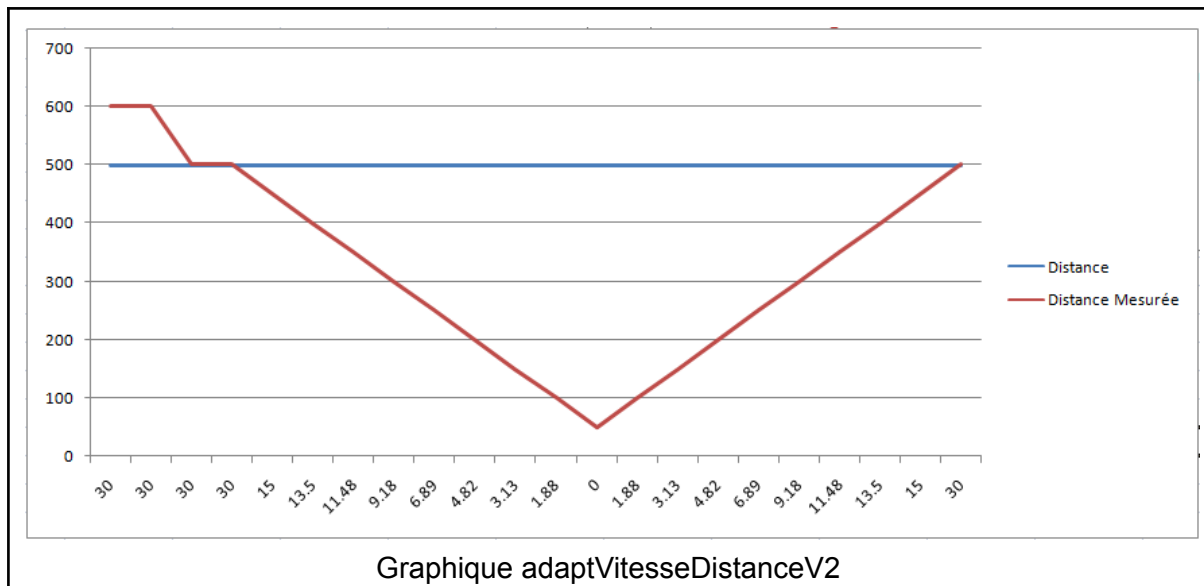
#### 3.4.1.1 : Exemple de la fonction de l'adaptation de vitesse :

Dans cet exemple on aura en distance choisie 5 m, soit 500 cm et une PWM de 30.



Graphique adaptVitesseDistance

Sur ce graphique on retrouve donc en **bleu** la distance choisie par l'utilisateur (5 m), et en **rouge** la différence entre la distance que l'on mesure et celle choisie par l'utilisateur. On voit bien que tant que l'on reste supérieur à la distance choisie rien ne se passe et la pwm reste stable à 30. Cependant si la distance vient à se réduire cela signifie que l'objet devant nous se rapproche et qu'il faut donc ralentir. A mesure que la distance se réduit la PWM aussi jusqu'à ce que l'on arrive à une distance inférieure à 80cm ou on passe la PWM à 0 pour s'arrêter.



Sur ce graphique on reprend l'exemple du dessus, cependant après avoir été à l'arrêt, l'objet devant nous se remet en route et s'éloigne, et la PWM peut donc remonter jusqu'à arriver à la PWM choisie par l'utilisateur.

### 3.4.2 : Fonction pour le mode auto :

La fonction **modeAuto** prend en paramètres 2 entiers, le premier correspond à la valeur retourner par le capteur Lidar ( La distance qu'il mesure ) et le second est la distance choisie par l'utilisateur à partir de laquelle on va ralentir. Le mode auto gèrera tout seul la PWM. On retrouve dans la fonction du mode auto certaine partie de la fonction **adaptVitesseDistance**. Dans cette fonction, la PWM passe à 0 si la distance passe en dessous de 50 cm, sinon on passe au calcul de la PWM. On retrouve au début de la boucle **else** une première condition **if** qui permet de relancer la PWM si elle était passée à 0 pour un arrêt. Ensuite on a une boucle **if** dans laquelle on rentrera si la distance devant nous est suffisante, sinon on passera dans la condition suivante afin de ralentir progressivement.

```
void modeAuto(int distanceLidar, int distanceChoisie)
{
    /* ADAPTER PWM EN FCT DE LA DISTANCE CHOISIE PAR L'UTILISATEUR */
    /* PREND EN COMPTE SI DEVANT CA ACCELERE DONC DISTANCELIDAR AUGMENTE */
    /* OU BIEN SUR LA DISTANCELIDAR REDUIT DONC CA FREINE DEVANT */

    double nouvVitesse = 0;
    double diffDistance = 0;
    int lock = 0;
    int multiplicateur;
    if( distance <= 50)
    {
        /*----ARRET----*/
        htim3.Instance->CCR1 = 0;
        lock = 0;
    }
    else{
        /*----POUR LA RELANCE----*/
        if(distanceLidar > 50 && htim3.Instance->CCR1 == 0 && lock == 0)
```

```

{
    htim3.Instance->CCR1 = 10;
    lock = 1;
}
if(distanceLidar > distanceChoisie )
{
    /*-- SI ON A UNE DISTANCE SUFFISANTE ALORS ON PEUT ACCELERER --*/
    if(htim3.Instance->CCR1 < 100)
    {
        /*----CALCUL POUR REDUIRE LA VITESSE----*/
        diffDistance = distanceLidar - distanceChoisie ;
        /*----ATTENTION SI ECART TROP IMPORTANT VAL NEGATIVE----*/
        /*----GESTION EN CAS DE VALEURS NEGATIVES----*/
        if(diffDistance > 5 && diffDistance < 100)
        {
            nouvVitesse = 1+(diffDistance/100);
            multiplicateur = 10;

        }
        else if(diffDistance > 100 && diffDistance < 1000 )
        {
            nouvVitesse = 1+(diffDistance/1000);
            multiplicateur = 50;

        }
        else if(diffDistance > 1000)
        {
            nouvVitesse = 1+(diffDistance/10000);
            multiplicateur = 75;

        }
        nouvVitesse = (multiplicateur*nouvVitesse);
        htim3.Instance->CCR1 = nouvVitesse;

    }
}
else if(distanceLidar < distanceChoisie )
{
    /*---- CAS OU ON EST PLUS PROCHE QUE PREVUS ----*/
    if(htim3.Instance->CCR1 > 0)
    {
        /*----CALCUL POUR REDUIRE LA VITESSE----*/
        diffDistance = distanceChoisie - distanceLidar;
        /*----ATTENTION SI ECART TROP IMPORTANT VAL NEGATIVE----*/
        /*----GESTION EN CAS DE VALEURS NEGATIVES----*/
        if(diffDistance > 100)
        {
            nouvVitesse = 1-(diffDistance/1000);

        }
        else if(diffDistance > 1000)
        {
            nouvVitesse = 1-(diffDistance/10000);

        }
        else{
            nouvVitesse = 1-(diffDistance/100);

        }
        nouvVitesse = (htim3.Instance->CCR1*nouvVitesse);
        htim3.Instance->CCR1 = nouvVitesse;

    }
}
}
}

```

### 3.4.3 : Fonction pour le calcul des tours/min :

Grâce à l'encodeur de notre moteur à courant continu on peut calculer sa vitesse de rotation, et celle en sortie du réducteur. On sait donc que pour que notre axe à la sortie du réducteur fasse un tour, l'axe du moteur doit faire 172 tours. On sait aussi que notre encodeur compte 48 fois par tour d'axe du moteur. On a donc  $172 \times 48 = 8256$ . Pour faire un tour, on aura compté 8256 fois.

La fonction **mesTrMin** prend donc en entrée un entier **tim4\_cnt** venant du **main**. Cette valeur s'incrémente à chaque fois que l'axe du moteur fait un tour.

```
tim4_cnt = (htim1.Instance->CNT / 24 );
```

On divise par 24 car on récupère ici la valeur sur 1 pin ([Voir schéma](#)), on aura donc 24 valeurs au lieu des 48. Lorsqu'on aura fait un tour on aura donc 24 valeurs que l'on divisera par 24 ce que fera 1, donc on aura fait 1 tour. Cette valeur sera passée en paramètre de ma fonction **mesTrMin**.

```
rpm = mesTrMin(tim4_cnt);
```

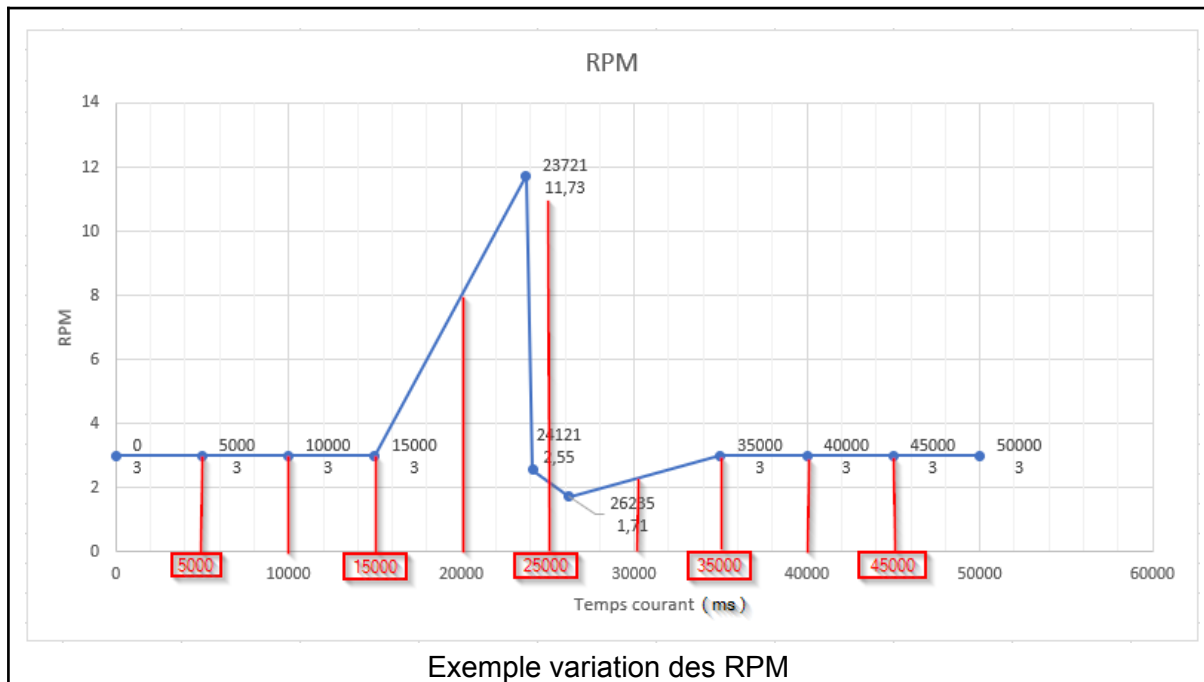
Dans la fonction **mesTrMin** on regarde ensuite quand la valeur est supérieure à 43 ( On a choisi ici 43 et non 48 pour compenser l'élan ). Lorsqu'on a détecté 1 tour on calcule donc le temps mis entre ce tour et le tour précédent. On récupère ensuite le nouveau temps pour calculer pour le prochain tour. Et on repasse le compteur de tour à 0 pour recompter un nouveau tour. On retourne ensuite le résultat.

```
double mesTrMin(int tim4_cnt)
{
    /*-----FONCTION QUI RETOURNERA LE NOMBRE DE TOURS PAR MINUTES-----*/
    /*-----ON PREND EN ENTREE LE COMPTEUR DE TOUR DU MOTEUR-----*/
    /*-----EXEMPLE AVEC DES VALEURS SIMPLES-----*/
    /*-----ON DIVISE NOTRE VALEUR EN ENTREE PAR 24 CAR MESURE SUR 1 PIN-----*/

    if(tim4_cnt >= 43)
    {
        /*----DEMI-TOUR----*/
        /*----MULTIPLIER TEMPS PAR 2----*/
        rpm = 60000/( (tpsCourant - montemps)*4 );
        montemps = tpsCourant;
        htim1.Instance->CNT = 0;
    }
    return rpm;
}
```



### 3.4.3.1 : Graphique variation des RPM :



Temps Courant	Mon temps	RPM
5000	0	3
10000	5000	3
15000	10000	3
20000	15000	3
25000	23721	11,73
30000	24121	2,55
35000	26235	1,71
40000	35000	3
45000	40000	3
50000	45000	3
55000	50000	3

Données utilisées pour le graphique

La fonction **mesTrMin** utilise donc le temps courant et un autre temps qui change à chaque fois que l'axe du moteur fait un tour. On voit sur le graphique que plus le temps entre le temps courant et mon temps est proche plus les RPM sont élevés, et a contrario plus le temps entre le temps courant et mon temps est grand plus les RPM sont bas.

- $25000 - 23721 = 1279$  ms d'écart  $\rightarrow 11,73$  RPM
- $30000 - 24121 = 5879$  ms d'écart  $\rightarrow 2,55$  RPM
- $35000 - 26235 = 8765$  ms d'écart  $\rightarrow 1,71$  RPM

### 3.5 : Main du programme :

Dans le main du programme on retrouvera l'initialisation des différentes variables qui seront utilisées dans le programme. On retrouve aussi des tests afin de s'assurer que l'écran LCD et le capteur Lidar soient bien connectés. Il y a aussi l'appel des interfaces pour le démarrage.

```
/*-----VERIFICATION CONNECTION LCD ET INIT-----*/
if(lcd_isConnected() == 1)
{
    /*CLEAR LCD*/
    lcd_clear();
    /*INIT LCD*/
    lcd_init();
    HAL_Delay(2000);
    lcd_clear();
}
lcd_maVoiture();
lcd_clear();
/*-----FIN DE LA PARTIE SUR LE LCD-----*/

/*-----INIT PWM POUR CONTROLLEUR MOTEUR-----*/
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
/*-----FIN INIT PWM POUR CONTROLLEUR MOTEUR-----*/

/*-----INIT DU CAPTEUR LIDAR-----*/
lidar_init();
/*-----FIN INIT DU CAPTEUR LIDAR-----*/
```

Ensuite on entre dans une boucle **while** qui va nous permettre de faire fonctionner notre programme sans durée de temps. Dans cette boucle on enregistrera le temps courant dans notre variable pour nous en servir pour nos différentes fonctions, on récupérera aussi les valeurs venant de l'encodeur, et du capteur lidar pour ensuite utiliser ces données dans nos fonctions. On appellera aussi **lcd\_Navig()** qui sera notre interface et qui nous permettra de naviguer dans les différentes interfaces.

```
while (1)
{
    /*-----INIT TEMPS COURANT-----*/
    tpsCourant = HAL_GetTick();

    /*-----VALEUR DU CAPTEUR DU MOTEUR-----*/
    /*-----DIVISION PAR 24 CAR ON A 24 MESURES POUR 1/2 TOUR DU MOTEUR-----*/
    /*-----IL FAUT 172 TOUR MOTEUR POUR UN TOUR DE L'ARBRE REDUCTEUR-----*/
    tim4_cnt = (htim1.Instance->CNT / 24 );
    rpm = mesTrMin(tim4_cnt);

    /*-----INTERFACE PRINCIPALE-----*/
    lcd_Navig();
}
```

```

/*----VERIFICATION CONNECTION LIDAR----*/
if(HAL_I2C_IsDeviceReady(&hi2c1,LIDAR_ADD,2,500)== HAL_OK)
{
    isConnected = 1;
    cmd[0] = 0x04;
    if(HAL_I2C_Mem_Write(&hi2c1,LIDAR_ADD,0x00,1,cmd,1,100) == HAL_OK)
    {
        //isConnected = 2;
        /*Lire les 2 octets venant de 0x8f*/
        cmd[0] = 0x8f;
        HAL_I2C_Master_Transmit(&hi2c1,LIDAR_ADD,cmd,1,100);
        HAL_I2C_Master_Receive(&hi2c1, LIDAR_ADD, data,2,100);
        distance =(data[0]<<8)|(data[1]);
    }
}
else{
    isConnected = -1;
}
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
}

```

## 4 : Tests unitaires et d'intégration :

### 4.1 : Test pour le capteur lidar :

Au lancement du programme on retrouve un test qui vérifie si le capteur Lidar est bien branché, si ce n'est pas le cas alors une variable passera à -1. Par défaut au lancement la variable est à 0. Si le capteur est bien branché alors cette variable passe à 1.

```
int isConnected;
```

Vérification de la connexion du capteur lidar :

```

if(HAL_I2C_IsDeviceReady(&hi2c1,LIDAR_ADD,2,500)== HAL_OK)
{
    isConnected = 1;
    /*----CODE----*/
}

else
{
    isConnected = -1;
}

```

On peut vérifier la variable **isConnected** dans les live expressions de Cubelde. Si on débranche le capteur pendant que le programme tourne, la variable **isConnected** passera à -1, si on le rebranche ensuite elle passera à 1.

<div> <div>(x)= isConnected</div> <div>int</div> <div>-1</div> </div> <p>Déconnecté</p>	<div> <div>(x)= isConnected</div> <div>int</div> <div>1</div> </div> <p>Connecté</p>
---	--

## 4.2 : Test pour l'écran LCD :

Afin de vérifier la connexion de l'écran lcd à la carte STM32, on a une fonction dans la librairie **i2c-lcd** qui regarde si notre écran est bien connecté.

```
int lcd_isConnected(void)
{
    int nblcd = 0;
    if(HAL_I2C_IsDeviceReady(&hi2c1,LCD_ADD, 2,500)== HAL_OK)
    {
        nblcd = 1;
    }
    return nblcd;
}
```

La fonction retourne donc **0** si l'écran LCD n'est pas connecté, et 1 si l'écran est bien connecté.

## 4.3 : Test pour les boutons :

Pour ce qui est du test des boutons, cela à été fait physiquement en fonction des différentes interfaces. Différents scénarios ont été exécutés.

### 4.3.1 : Scénario pour l'interface principale :

Sur l'interface principale, l'utilisateur peut utiliser les 3 boutons.

- Bouton 1 → Changer la distance choisie
- Bouton 2 → Changer la vitesse choisie
- Bouton 3 → Changer d'interface

Les différents tests effectués sur les bouton pour cette interface sont :

- Appui sur le bouton 1 et 2 en même temps
- Appui sur le bouton 2 et 3 en même temps
- Appui sur le bouton 1 et 3 en même temps
- Appui sur les 3 boutons en même temps

### 4.3.2 : Scénario pour l'interface d'affichage de la distance et des RPM :

Sur l'interface qui affiche les informations reçue par les différents capteur seul le bouton 3 est actif et il permet de revenir à l'interface principale.

Les différents tests effectués sur cette interface sont :

- Appui sur le bouton 1 et vérification que rien n'a changé sur l'interface principale

- Appui sur le bouton 2 vérification que rien n'a changé sur l'interface principale
- Appui sur le bouton 1 et 2 en même temps
- Appui sur le bouton 2 et 3 en même temps
- Appui sur le bouton 1 et 3 en même temps

#### **4.3.3 : Scénario pour l'interface de saisie de la distance :**

Sur l'interface pour la saisie de la distance, les 3 boutons sont utilisés.

- Bouton 1 → Se déplacer à gauche
- Bouton 2 → Valider
- Bouton 3 → Se déplacer à droite

Les différents tests effectués sur les bouton pour cette interface sont :

- Appui sur le bouton 1 et 2 en même temps
- Appui sur le bouton 2 et 3 en même temps
- Appui sur le bouton 1 et 3 en même temps
- Appui sur les 3 boutons en même temps

#### **4.3.4 : Scénario pour l'interface du mode auto :**

Dans cette interface le seul bouton qui sera utilisé sera le bouton 2 qui permettra de valider son choix et de quitter le mode auto.

Les différents tests effectués sur cette interface sont :

- Appui sur le bouton 1 et vérification que rien n'a changé sur l'interface principale
- Appui sur le bouton 2 vérification que rien n'a changé sur l'interface principale
- Appui sur le bouton 1 et 2 en même temps
- Appui sur le bouton 2 et 3 en même temps
- Appui sur le bouton 1 et 3 en même temps

#### **4.3.5 : Scénario pour les interfaces de transitions :**

Pour les interfaces de transitions les boutons 1 et 2 seront utilisés.

- Bouton 1 → Choisir "**OUI**" ou "**NON**"
- Bouton 2 → Valider le choix

Les différents tests effectués sur cette interface sont :

- Appui sur le bouton 1 et 2 en même temps
- Appui sur le bouton 2 et 3 en même temps
- Appui sur le bouton 1 et 3 en même temps
- Appui sur les 3 boutons en même temps

## **5 : Conclusion :**

Le but de ce projet était de réaliser une application afin de récupérer à intervalle régulier les données d'un Lidar et ensuite de programmer l'asservissement d'un moteur à courant continu. Le programme réalisé ici permet de récupérer les données du capteur Lidar et de l'encodeur du moteur afin de connaître la distance et la vitesse de rotation du moteur. De plus, un écran LCD et 3 boutons ont été ajoutés afin de permettre une interaction avec l'utilisateur. L'asservissement du moteur en fonction de la distance retournée par le capteur Lidar a donc été fait.