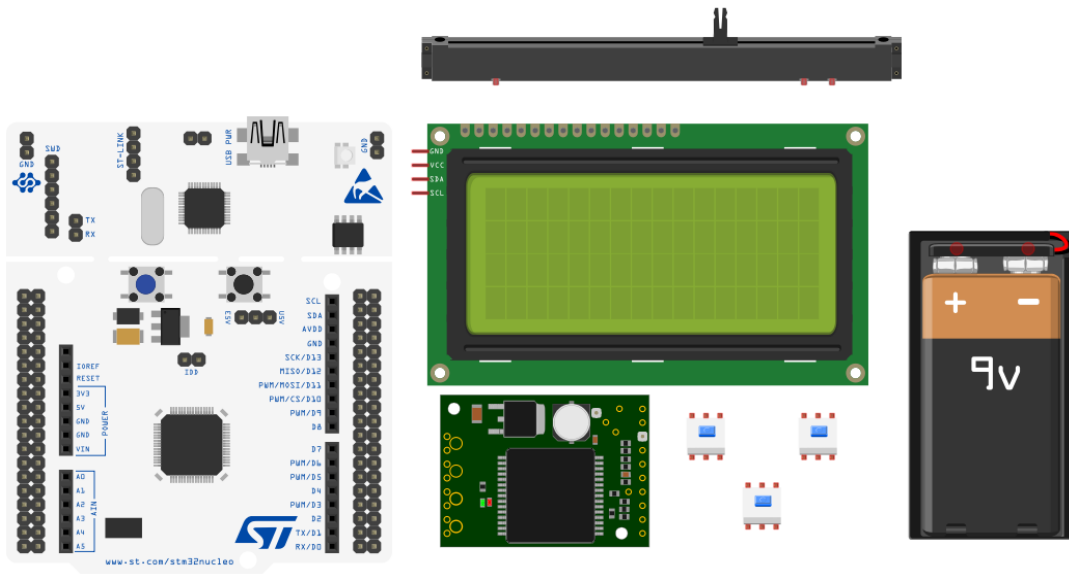


# Pilotage d'une commande d'axe



Compte rendu de projet

Albouy Yann

Version	Date
1.0	08/02/2021
1.1	04/03/2021
1.2	07/03/2021

## **Sommaire :**

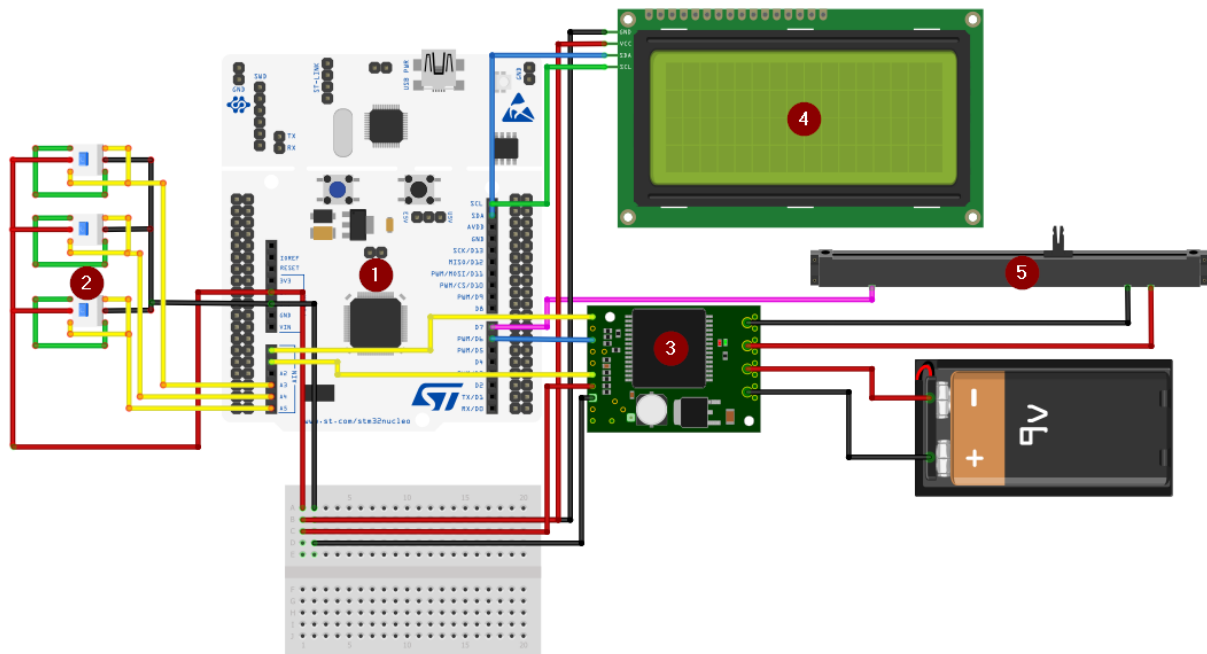
<b>1 : Sujet du projet :</b>	<b>4</b>
<b>2 : Matériel utilisé :</b>	<b>4</b>
2.1 : Carte STM32F411RE :	5
2.1.1 Ports utilisés :	5
2.2 : Bouton poussoir :	6
2.3.1 : Utilisation des boutons pour l'interface principale :	6
2.3.2 : Utilisation des boutons pour l'interface d'impression :	6
2.3.3 : Branchement du bouton :	7
2.3 : Contrôleur moteur pololu md15a :	8
2.4 : Ecran Clear Display CLCD 420 :	9
2.4.1 : Commande de l'écran LCD :	9
2.4.2 : Utilisation des sous pixels :	10
2.5 : Potentiomètre à glissière motorisé :	10
<b>3 : Programme :</b>	<b>11</b>
3.1 : Tâche de récupération des données :	11
3.1.1 : Code de la tâche de récupération des données :	12
3.2 : Tâche de déplacement du potentiomètre :	12
3.2.1 : Sens de rotation :	13
3.2.2 : Code de la tâche :	13
3.3 : Tâche pour l'écran LCD :	13
3.3.1 : Code de la tâche :	14
3.4 : Librairie pour les boutons :	14
3.4.1 : Exemple pour le bouton de validation :	15
3.4.2 : Utilisation des boutons :	16
3.5 : Librairie pour l'écran LCD :	16
3.5.1 : Interface lcd_Position() :	17
3.5.2 : Interface impression() :	17
3.5.3 : Commande pour l'écran LCD :	18
3.6 : Librairie pour imprimer :	18
<b>4 : Logiciel QT :</b>	<b>19</b>
4.1 : Fonctionnement logiciel :	19
4.2 : Ajout du dessin dans STM32Cube IDE :	20
4.2 : Interface du logiciel :	21
4.2.1 : Interface principale :	21
4.2.2 : Ouverture image :	21
4.2.3 : Enregistrement image :	22
4.2.4 : Image convertie :	22
<b>5 : Tests :</b>	<b>23</b>
5.1 : Test pour l'écran LCD :	23
5.2 : Test pour les boutons :	23
5.2.1 : Scénario pour l'interface principale :	23

5.2.2 : Scénario pour l'interface d'impression :	23
5.2.3 : Scénario pour la commande d'axe :	24
<b>6 : Conclusion :</b>	<b>24</b>

## 1 : Sujet du projet :

Le sujet du projet est ici de piloter une commande d'axe tout en affichant sa position en temps réel sur un écran LCD. Il devra y avoir une vérification de la position grâce à un retour capteur. La première partie de ce projet consistera donc à faire bouger la commande d'axe, qui est un potentiomètre à glissière motorisé, afin de le mettre à la position voulue. Dans un second temps, la glissière motorisé devra être capable de se déplacer grâce à des instructions stockées dans un tableau. Ainsi on pourra avoir un comportement proche de celui d'une imprimante, car la glissière devra se déplacer rapidement tout en gardant sa précision parce qu'elle devra se mettre à la bonne position. Le programme fonctionnera sur une carte avec un OS ( FreeRTOS ) STM32F411RE Nucleo. L'OS permettra ainsi de partager les ressources entre les différentes tâches que la carte devra exécuter.

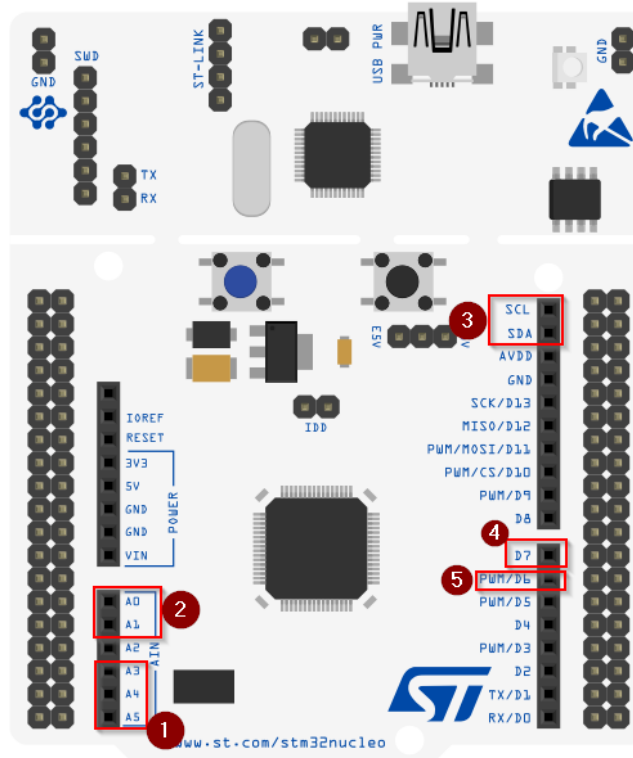
## 2 : Matériel utilisé :



1	<a href="#">Carte STM32F411RE</a>
2	<a href="#">Bouton Poussoir</a>
3	<a href="#">Contrôleur moteur Pololu md15a</a>
4	<a href="#">Ecran Clear Display CLCD 420</a>
5	<a href="#">Potentiomètre à glissière motorisé</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. Sur celle-ci on utilisera un OS temps réel appelé FreeRTOS, qui nous permettra d'avoir un scheduler ( Ordonnanceur ) et ainsi de mieux répartir les ressources entre les différentes tâches qui seront exécutées par la carte.



### 2.1.1 Ports utilisés :

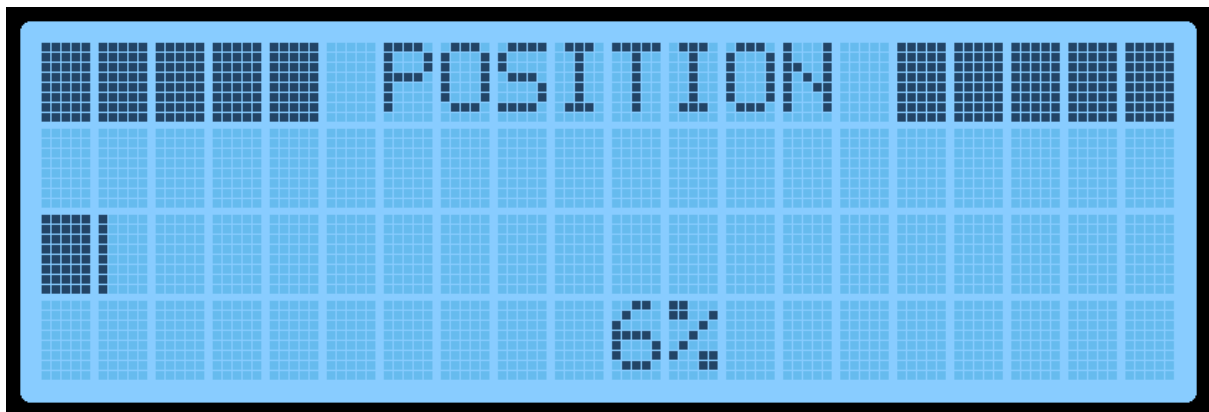
1 → Bouton 1	Utilise l'entrée GPIO A5
1 → Bouton 2	Utilise l'entrée GPIO A4
1 → Bouton 3	Utilise l'entrée GPIO A3
2 → Choix du sens contrôleur moteur	Utilise l'entrée GPIO A0
2 → Choix du sens contrôleur moteur	Utilise l'entrée GPIO A1
3 → D14 et D15 / SDA et SCL	Utilisés pour envoyer des commandes à l'écran LCD.
4 → Potentiomètre	Récupération de la valeur du potentiomètre
5 → Contrôle du moteur	Envoi de la PWM sur le port D6

## 2.2 : Bouton poussoir :



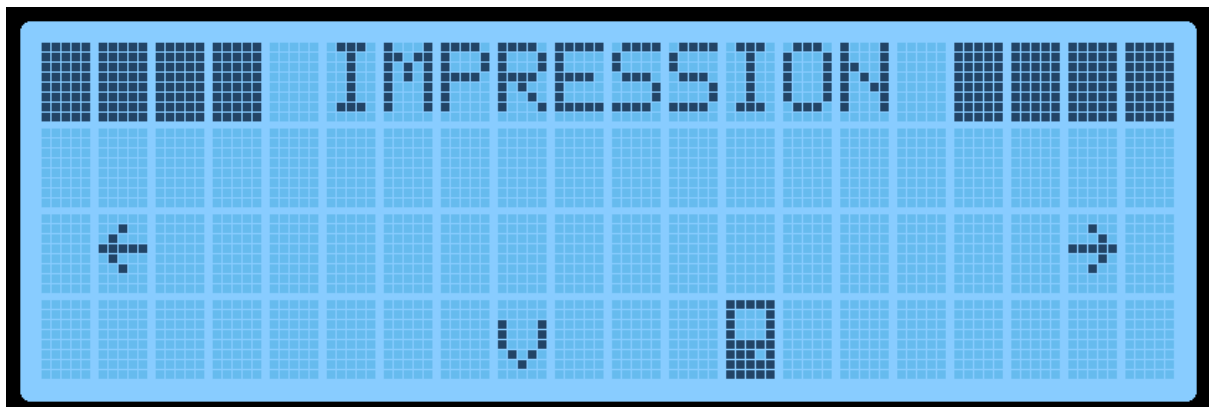
Afin de faciliter l'utilisation du programme et d'avoir une interaction avec l'utilisateur, le projet comporte 3 boutons SPPH410200 de chez Alps Alpine qui en fonction de l'interface auront différentes fonctions. Le premier bouton est branché en GPIO IN sur le port PC0, le second bouton est branché en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

### 2.3.1 : Utilisation des boutons pour l'interface principale :







L'interface principale permet de visualiser en temps réel la position du potentiomètre à glissière. On retrouve sa position sous forme de pourcentage. Le **bouton 1** permet de déplacer le potentiomètre vers la gauche, tandis que le **bouton 2** permet de déplacer le potentiomètre vers la droite. Le **bouton 3** lui sert à changer d'interface pour aller sur l'interface d'impression.

### 2.3.2 : Utilisation des boutons pour l'interface d'impression :

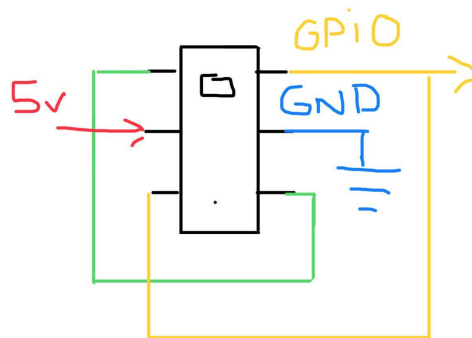


L'interface d'impression utilise les 3 boutons. Le **bouton 1** permet de déplacer le curseur vers la gauche, le **bouton 3** permet quant à lui de déplacer le curseur vers la droite. Le

**bouton 2** lui sert à valider le choix fait par l'utilisateur afin de lancer l'impression, de changer de dessin, ou bien de revenir à l'interface principale.

	Cette touche permet de changer le dessin que l'on souhaite imprimer
	Cette touche permet de changer le dessin que l'on souhaite imprimer
	Cette touche permet de sortir de l'interface d'impression et de revenir à l'interface principale sans lancer d'impression.
	Permet de lancer l'impression du dessin sélectionné à l'aide des flèches.

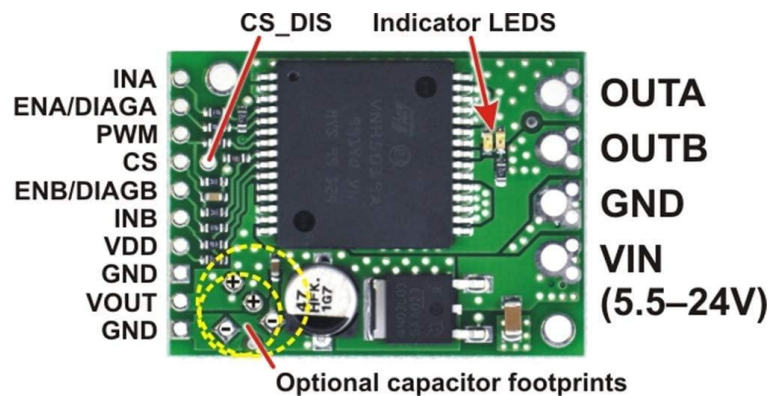
### 2.3.3 : Branchement du bouton :



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

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

## 2.3 : Contrôleur moteur pololu md15a :

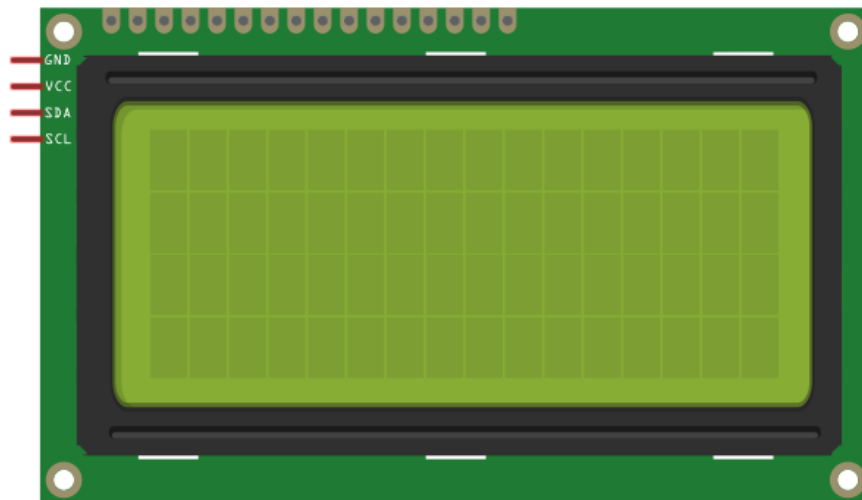


Ce composant permet d'envoyer une PWM à notre moteur pour contrôler sa vitesse de rotation, grâce à la PWM qu'on lui envoie la vitesse de rotation varie. Il permet aussi de choisir le sens de rotation de notre moteur ( sens des aiguilles d'une montre, sens inverse des aiguilles d'une montre ).

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
PWM	Brancher sur le port D5 qui envoie la PWM
INA	Brancher sur le GPIO A0 de la carte permet de contrôler le sens dans lequel le moteur tournera. Lorsque INA sera actif, le sens de rotation sera le sens des aiguilles d'une montre.
INB	Brancher sur le GPIO A1 de la carte permet de contrôler le sens dans lequel le moteur tournera. Lorsque INB sera actif, le sens de rotation sera le sens inverse des aiguilles d'une montre.



## 2.4 : Ecran 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.

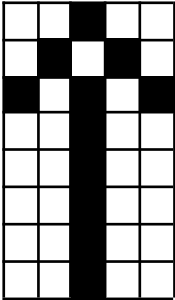
### 2.4.1 : Commande de l'écran LCD :

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.

	<p>Ensuite pour chaque ligne on vas faire la conversion en Hexa :</p> <p><b><u>Exemple pour la troisième ligne :</u></b></p> <table border="1" data-bbox="448 546 1375 674"><tr><th>16</th><th>8</th><th>4</th><th>2</th><th>1</th></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table> <p>La valeur à saisir pour la commande de la troisième ligne sera : 0x15</p>	16	8	4	2	1	1	0	1	0	1
16	8	4	2	1							
1	0	1	0	1							

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 : Potentiomètre à glissière motorisé :



Le potentiomètre à glissière motorisé Bourns est composé de 3 éléments :

- Un potentiomètre à glissière de 10k
- Un moteur à courant continu
- Une courroie pour le déplacement de la glissière

Le composant comporte 7 pins, et on se servira seulement de 3 pins.

Pin n°1 → Ground du potentiomètre
Pin n°2 → Courant +5v pour alimenter le potentiomètre
Pin n°3 → Récupération de la valeur du potentiomètre

Le moteur du potentiomètre à glissière comporte quant à lui seulement 2 fils, un **Ground** et un **+5V** afin de l'alimenter.

### **3 : Programme :**

Dans ce projet, la carte fonctionne avec un OS ( FreeRTOS ), on est donc plus sur un programme Bare-métal ( Sans OS ) comme le projet précédent. FreeRTOS est un OS temps réel, ce qui signifie que les actions qu'il réalisera seront faites à un temps déterminé quoiqu'il se produise.

L'os permet aussi d'ajouter un niveau d'abstraction supplémentaire par rapport à de la simple programmation C, car il ajoute un **“scheduler”** ou **ordonnanceur** qui permet d'avoir des notions de priorités pour des tâches. Les tâches sont des programmes dans l'OS qui possèdent leurs propres droits, elles ont un point d'entrée, et sont ensuite exécutées dans une boucle infinie. Chaque tâche a sa propre pile, ainsi que la copie de toutes les variables locales à la tâche. Pour ce projet l'ordonnancement en **tourniquet** ou **Round-Robin** a été choisi. Celui-ci permet de partager équitablement le temps processeur entre chacun des processus, avec un intervalle de temps bien précis. La performance de l'ordonnanceur dépendra de notre intervalle de temps (Quantum) choisi.

#### **3.1 : Tâche de récupération des données :**

La tâche de récupération des données permet de venir lire sur le port D7 et convertir la valeur retournée par le potentiomètre. La valeur que nous retourne le capteur est comprise entre 1 et 4096, afin d'avoir un pourcentage précis d'afficher sur l'écran on multiplie donc la valeur que l'on reçoit par 100, puis on la divise par 4096 ( **$(uiAnalogData * 100) / 4096$** ). Ainsi on obtient un pourcentage précis qui correspond à la position de la glissière. Cette valeur est ensuite transmise à l'écran LCD pour que l'utilisateur puisse voir en temps réel où se trouve le capteur.

En faisant bouger la glissière avec un pas de 1% par 1%, on incrémente donc la valeur de notre potentiomètre de 0,024414063.

Calcul pour le 1er pourcent	$(1 * 100) / 4096$	0,024414063
Calcul à 50%	$((50 * 0,024414063) * 4096) / 100$	50,000001024
Calcul à 100%	$((100 * 0,024414063) * 4096) / 100$	100,000002048

On voit donc bien qu'en appuyant 50 fois sur le bouton pour mettre la glissière à 50%, celle-ci s'y placera précisément, de même pour 100% en appuyant 100 fois.

### 3.1.1 : Code de la tâche de récupération des données :

```
void StartTask03(void const * argument)
{
    /* USER CODE BEGIN StartTask03 */
    /* Infinite loop */
    for(;;)
    {
        /*----Recuperation de La position du potentiometre----*/
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1,1000);
        uiAnalogData = HAL_ADC_GetValue(&hadc1);
        percent = (uiAnalogData *100)/4096;
        itoa(percent,mot,10);
        lengthTab = log10(percent) + 1;
        HAL_ADC_Stop(&hadc1);
        osDelay(1);
    }
    /* USER CODE END StartTask03 */
}
```



### 3.2 : Tâche de déplacement du potentiomètre :

Cette tâche permet de déplacer la glissière en envoyant une PWM sur le port D6. Par défaut, la tâche remet la glissière à la position 0. En fonction du bouton pressé par l'utilisateur, la glissière se déplacera à gauche ou bien à droite, ce qui correspond à une incrémentation ou bien une décrémentation de **percent**. De plus l'ordonnanceur en Round robin nous permet d'avoir une vérification en continue du placement de la commande d'axe. Dans la tâche on retrouve 2 cas :

pushMouvement < percent	Dans ce cas, on va décrétement percent car à chaque appui sur <b>pushMouvement</b> , sa valeur se décrémente et percent fait en sorte de la suivre. Pour cela on fait tourner le moteur dans le bon sens en modifiant les valeurs des GPIO A0 et A1, et ensuite on envoie la PWM pour faire bouger la glissière.
pushMouvement > percent	Dans ce cas, on va incrémenter percent car à chaque appui sur <b>pushMouvement</b> , sa valeur s'incrémente et percent fait en sorte de la suivre. Pour cela on fait tourner le moteur dans le bon sens en modifiant les valeurs des GPIO A0 et A1, et ensuite on envoie la PWM pour faire bouger la glissière.

### 3.2.1 : Sens de rotation :

En fonction de la valeur des GPIO, le sens dans lequel tournera le moteur sera différent.

	
<pre>HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, 1); HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, 0);</pre>	<pre>HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, 0); HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, 1);</pre>

### 3.2.2 : Code de la tâche :

```
void StartTask02(void const * argument)
{
    /* USER CODE BEGIN StartTask02 */
    /* Infinite Loop */
    for(;;)
    {
        HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_1);
        /*----Tache Synchronisation Mouvement et Push----*/
        if(pushMouvement == percent)
        {
            /*----On est pas encore arriver----*/
            htim3.Instance->CCR1 = 0;
        }
        else if(pushMouvement < percent)
        {
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, 1);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, 0);
            htim3.Instance->CCR1 = 10;
        }
        else if(pushMouvement > percent)
        {
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, 0);
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, 1);
            htim3.Instance->CCR1 = 10;
        }
        osDelay(1);
    }
    /* USER CODE END StartTask02 */
}
```

### 3.3 : Tâche pour l'écran LCD :

Grâce à cette tâche, on peut afficher les informations utiles pour l'utilisateur sur l'écran. On retrouve dans cette tâche les différentes interfaces qui sont accessibles pour l'utilisateur, ainsi que la vérification de connexion de l'écran LCD sur l'I2C. Les fonctions **impression()** et **lcd\_Position()** viennent de la librairie **i2c\_lcd.c**. Comme la tâche "**tacheEcranLCD**" permet d'avoir accès aux dessins, la taille de sa pile a dû être augmentée car les tableaux de

coordonnées ont une taille conséquente en fonction du dessin. Elle à donc été passé de 128 à 512 mots

### 3.3.1 : Code de la tâche :

```
void tacheEcranLCD(void const * argument)
{
    /* USER CODE BEGIN tacheEcranLCD */

    /*----INIT DE L'ECRAN LCD----*/
    lcd_clear();
    /*-----FIN DE CONFIGURATION LCD-----*/
    /* Infinite loop */
    for(;;)
    {
        /*----VERIFIE CONNEXION LCD SUR I2C----*/
        if(HAL_I2C_IsDeviceReady(&hi2c1,LCD_ADD, 2,500)== HAL_OK)
        {
            nblcd = 1;
        }
        if(pushInterface == 0)
        {
            refresh = 0;
            lcd_Position();
        }
        else if(pushInterface == 1)
        {
            /*----INTERDIRE UTILISATION CERTAINES FONCTIONS----*/
            onePushDec = 1;
            onePushInc = 1;
            onePushInterface = 1;
            impression();
        }
        else if(pushInterface >1)
        {
            pushInterface = 0;
        }
        osDelay(1);
    }
    /* USER CODE END tacheEcranLCD */
}
```

## 3.4 : 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.

```

/*
 * LibBouton.h
 *
 * Created on: Feb 15, 2021
 * Author: Yann
 */

#ifndef INC_LIBBOUTON_H_
#define INC_LIBBOUTON_H_

#include "stm32f4xx_hal.h"

/*----PASSER D'UNE INTERFACE A L'AUTRE----*/
void boutonInterface();

/*----PERMET DE DEPLACER LE CURSEUR VERS LA DROITE DANS L'INTERFACE IMPRESSION----*/
void boutonDroit();

/*----PERMET DE DEPLACER LE CURSEUR VERS LA GAUCHE DANS L'INTERFACE IMPRESSION----*/
void boutonGauche();

/*----DEPLACER LA GLISSIERE VERS 0----*/
void boutonDecrement();

/*----DEPLACER LA GLISSIERE VERS 100----*/
void boutonIncrement();

/*----VALIDER LE CHOIX DE L'UTILISATEUR----*/
void boutonValide(void);
#endif /* INC_LIBBOUTON_H_ */

```

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\*\*\*\*\***, ou **push\*\*\*\*\*** désignant la variable à incrémenter.

### 3.4.1 : Exemple pour le bouton de validation :

```

void boutonValide(void)
{
    /*LIRE LE PIN ET LE ONEPUSH2 POUR SAVOIR SI ON CLIC*/
    /*ON A DONC UNE DOUBLE CLE POUR EVITER L'APPUI CONTINU*/
    if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_0) == 1 && onePushValide == 0)
    {
        lcd_clear();
        onePushValide++;
        buttonValide = buttonValide +1;
    }
    else if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_0) == 0){
        onePushValide = 0;
    }
}

```

### 3.4.2 : 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 qui permet de modifier la position de la glissière.

```
#include "libBouton.h"

/*----Affichage de La position de La glissiere----*/
void lcd_Position()
{
    boutonInterface();
    boutonDecrement();
    boutonIncrement();
    . . .
}
```

### 3.5 : 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.

```
/*
 * i2c_lcd.h
 *
 * Created on: Feb 9, 2021
 * Author: Yann
 */

#ifndef INC_I2C_LCD_H_
#define INC_I2C_LCD_H_
#include "stm32f4xx_hal.h"

void lcd_clear(void);

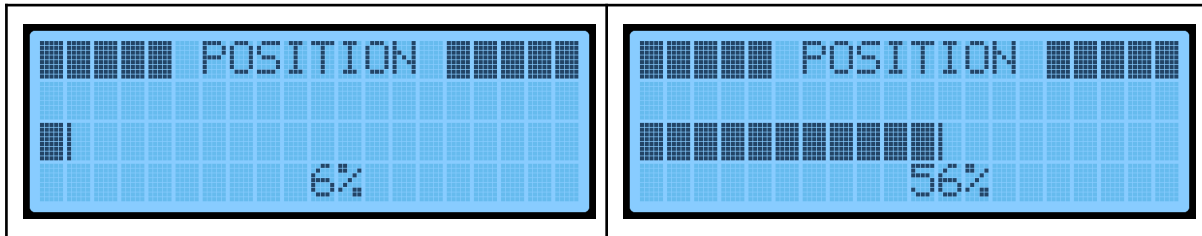
/*----Affichage de La position de La glissiere----*/
void lcd_Position();

/*----Interface que L'utilisateur verra pour L'impression----*/
void impression();
#endif /* INC_I2C_LCD_H_ */
```



### 3.5.1 : Interface `lcd_Position()` :

Cette fonction permet à l'utilisateur de visualiser en temps réel la position de la glissière. Grâce à cette fonction il a un retour visuel avec une barre d'avancement et le pourcentage qui représente la position. Lorsque l'utilisateur appuiera sur les boutons pour faire bouger la glissière l'interface se mettra instantanément à jour.

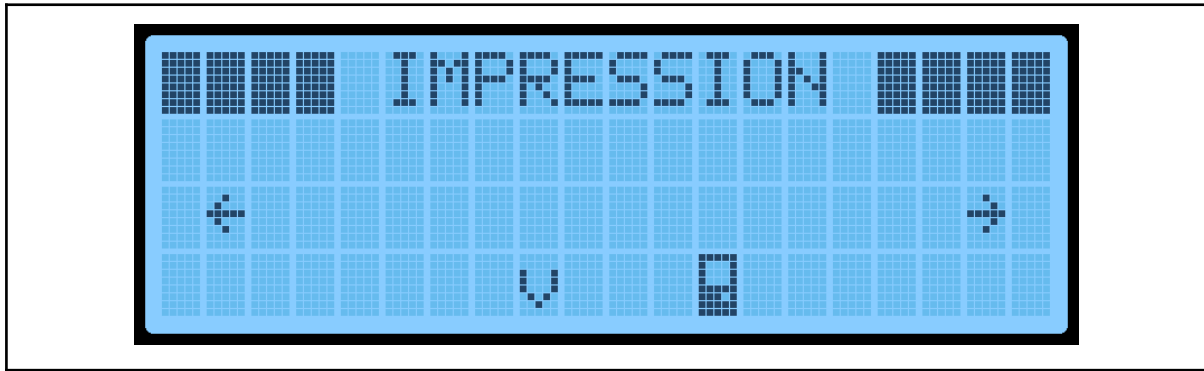


### 3.5.2 : Interface `impression()` :

L'interface impression permet à l'utilisateur de choisir quel dessin il veut imprimer. Grâce aux 3 boutons, il pourra changer de dessin, lancer une impression, ou bien revenir à l'interface principale.

```
if(buttonPosition == 0)
{
    /*----REVENIR VERS 1 POUR LE CHOIX DE L'IMPRESSION----*/
    nomDessinNb = nomDessinNb - 1;
    boutonValide = 0;
}
else if(buttonPosition == 1)
{
    /*----LANCER L'IMPRESSION----*/
    impressionDessin(nomDessinNb);
    boutonValide = 0;
    taille = 0;
}
else if(buttonPosition == 2)
{
    /*----QUITTER MODE IMPRESSION----*/
    boutonValide = 0;
    buttonPosition = 0;
    refresh = 0;
    pushInterface = 0;
    lcd_clear();
}
else if(buttonPosition == 3)
{
    /*----ALLER VERS +1 POUR LE CHOIX DE L'IMPRESSION----*/
    nomDessinNb = nomDessinNb + 1;
    boutonValide = 0;
}
```

S'il choisit de lancer l'impression, alors la fonction **impressionDessin()** sera lancée. Cette fonction vient lire les coordonnées du tableau qui a été choisi par l'utilisateur, et ainsi faire bouger la glissière aux positions stockées dans celui-ci.



### 3.5.3 : Commande pour l'écran LCD :

Dans la librairie pour l'écran LCD on retrouve aussi des fonctions qui envoient simplement des commandes afin de nettoyer l'écran.

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

### 3.6 : Librairie pour imprimer :

Cette librairie permet d'avoir une fonction qui stocke les coordonnées de nos dessins ( **stockDessin(int dessinsChoisis)** ). L'utilisateur peut avoir accès au dessin qu'il souhaite à l'aide d'un entier. On retrouve aussi la fonction ( **impressionDessin(int)** ) qui permet de faire bouger le potentiomètre à glissière grâce aux coordonnées venant de **stockDessin**.

```
/*
 * moteurImpression.h
 *
 * Created on: 14 févr. 2021
 * Author: Yann
 */

#ifndef INC_MOTEURIMPRESSION_H_
#define INC_MOTEURIMPRESSION_H_

#include "stm32f4xx_hal.h"

/*----Fonction qui vas faire le dessin----*/
void impressionDessin(int);
```

```

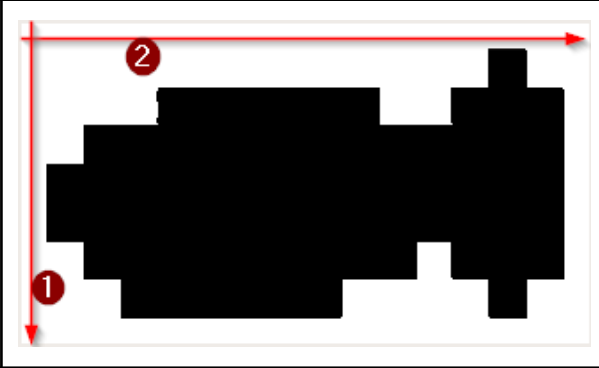
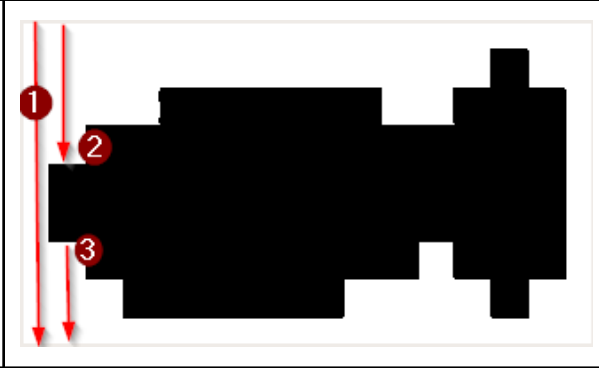
/*----Stockage des coordonnées des dessins----*/
int* stockDessin(int dessinsChoisis);
#endif /* INC_MOTEURIMPRESSION_H_ */

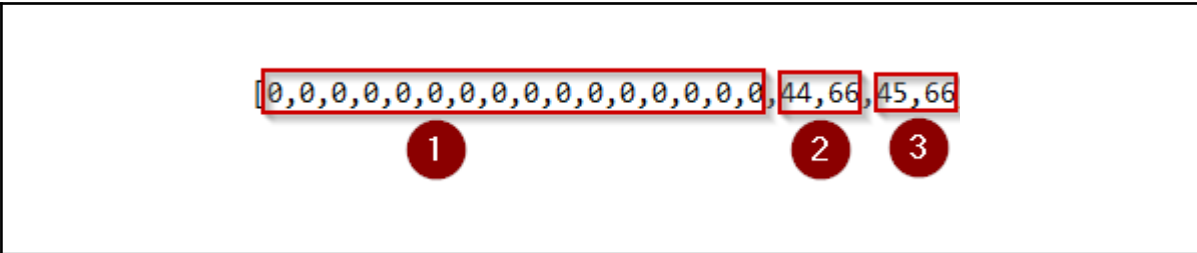
```

## 4 : Logiciel QT :

### 4.1 : Fonctionnement logiciel :

Afin d'obtenir un tableau contenant les coordonnées de notre image, l'utilisateur doit d'abord choisir une image au format souhaité ( png, jpg, jpeg ). Une fois celle-ci ouverte, le logiciel va la transformer en matrice de largeur 100 car notre potentiomètre à glissière varie entre 0 et 100. En changeant la largeur de l'image, la longueur va aussi être modifiée afin de garder le même ratio que l'image de base. On obtiendra donc une matrice représentant notre image qu'on va ensuite convertir en noir et blanc. Une fois notre image noir et blanc obtenue, on la parcourt dans la longueur afin de récupérer les coordonnées de nos pixels noirs. S'il n'y en a pas, alors on stockera dans un tableau à 1 dimension la valeur 0 pour la première coordonnée, et 0 pour la dernière coordonnée de la longueur. Par contre si on trouve un ou plusieurs pixels noir, on stockera la position ( Largeur ) du premier pixel noir, et du dernier.

	
<p>1 → Largeur 2 → Longueur</p>	<p>1 → Zone blanche donc coordonnées [0,0] 2 → Premier pixel noir 3 → Dernier pixel noir</p>


<p>1 → Coordonnées pixel blanc donc [0,0] 2 → On à des pixels noir donc on récupère les coordonnées de début [44] et de fin [66] 3 → On à des pixels noir donc on récupère les coordonnées du début [45] et de la fin [66]</p>

Notre tableau de coordonnées aura donc une taille de  $2 \times (\text{longueur de l'image})$ , car à chaque longueur on stocke la coordonnées du premier pixel noir, et la coordonnées du dernier pixel noir. Si il n'y en a pas on aura 0,0.

## 4.2 : Ajout du dessin dans STM32Cube IDE :

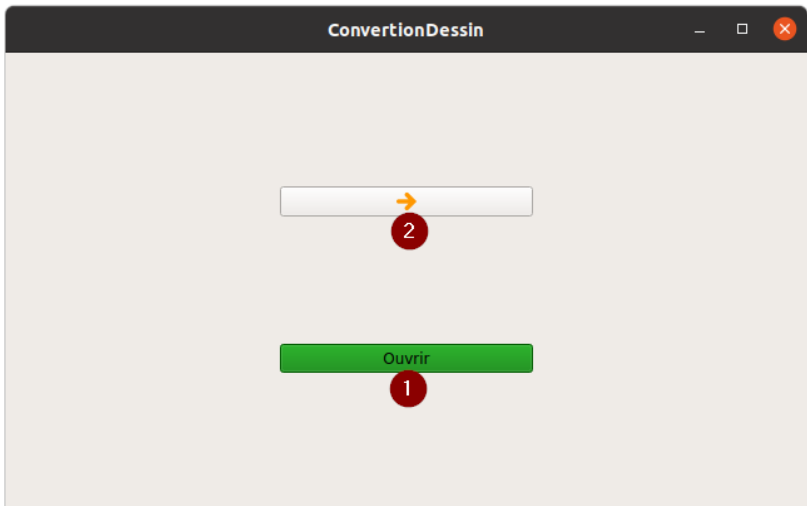
Afin de pouvoir imprimer des dessins à l'aide de la fonction **impressionDessin()** qui permet d'imprimer des dessins stockés dans un tableau, un logiciel en C++ a été fait pour convertir des images simples en tableaux de coordonnées. Le programme prend en entrée une image au format que l'on souhaite ( png, jpeg, jpg ) et nous sortira un fichier texte avec le tableau contenant les coordonnées pour imprimer le dessin. Ces coordonnées seront à mettre dans **moteurImpression.c** → **stockDessin()**. Dans cette fonction il suffira de rajouter nos coordonnées obtenues dans le tableau qui contient déjà les autres dessins :

- **( int [ ] ){ ...Coordonnées....},**

Ensuite dans **i2c\_lcd.c** on pourra incrémenter le **#define NBDESSINS** qui permet de connaître le nombre de dessins que comporte notre tableau venant de **stockDessin()**, et rajouter dans le tableau au début de **impression()** → **tabNom[][]** le nom du dessin qu'on vient de rajouter.

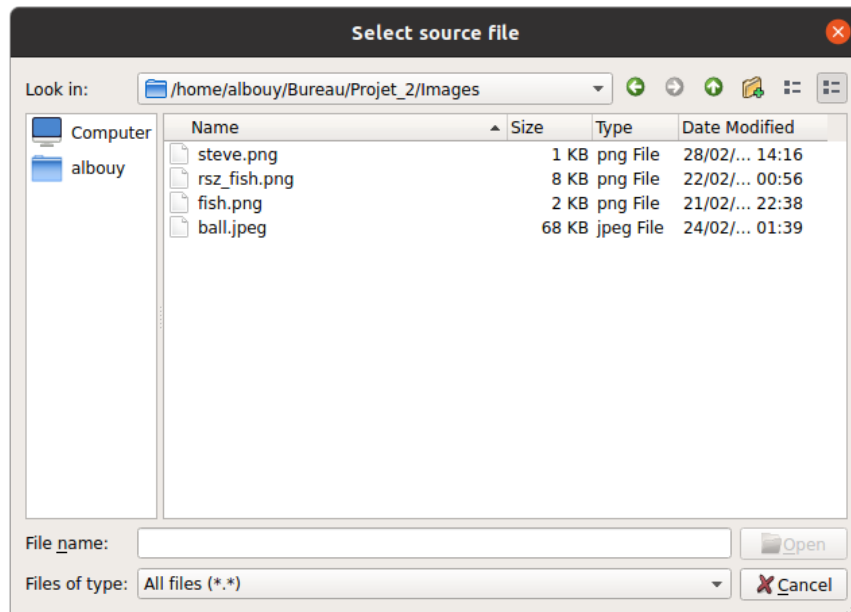
## 4.2 : Interface du logiciel :

### 4.2.1 : Interface principale :



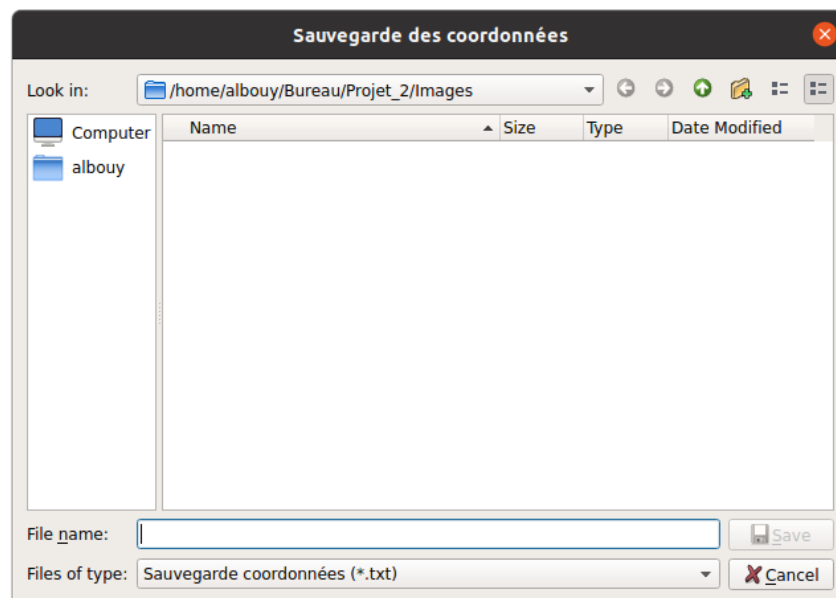
1 → Bouton pour ouvrir une image dans le format que l'on souhaite
2 → Bouton pour enregistrer et convertir notre image en coordonnées

### 4.2.2 : Ouverture image :



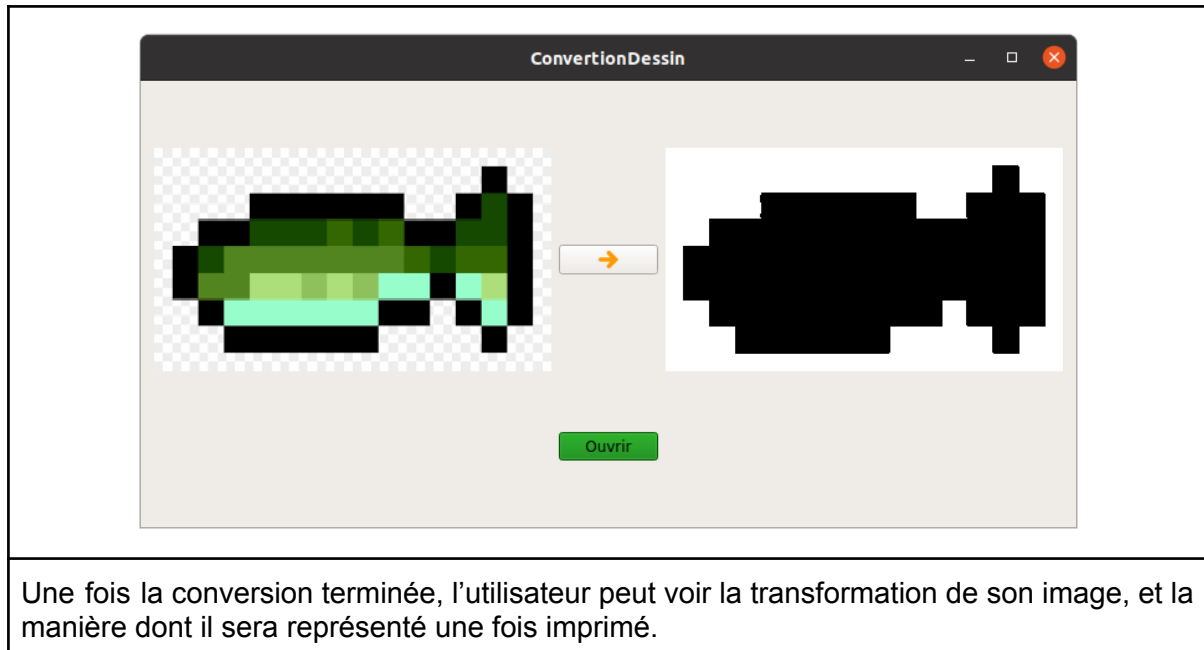
Lors du clic sur le **bouton 1**, la fenêtre suivante s'ouvre. Elle permet de choisir quelle image l'utilisateur va ouvrir afin de la convertir.

#### 4.2.3 : Enregistrement image :



Lors du clic sur le bouton 2, la fenêtre suivante s'ouvre et permet à l'utilisateur de choisir ou sera enregistrer le fichier **.txt** contenant les coordonnées de l'image.

#### 4.2.4 : Image convertie :



## 5 : Tests :

### 5.1 : Test pour l'écran LCD :

Afin de vérifier la connexion de l'écran LCD à la carte STM32F411RE, on a un test au début de la **tacheEcranLCD** qui regarde si notre écran est bien connecté.

```
/*-----VERIFIE CONNEXION LCD SUR I2C-----*/  
if(HAL_I2C_IsDeviceReady(&hi2c1,LCD_ADD, 2,500)== HAL_OK)  
{  
    nblcd = 1;  
}
```

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

### 5.2 : Test pour les boutons :

Pour tester les boutons, cela a été fait physiquement en fonction des différentes interfaces. Plusieurs scénarios ont donc été exécutés.

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

Sur cette interface on peut utiliser les 3 boutons :

- Bouton 1 → Déplacer la commande d'axe
- Bouton 2 → Déplacer la commande d'axe
- 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

### **5.2.2 : Scénario pour l'interface d'impression :**

Sur l'interface d'impression, l'utilisateur peut utiliser les 3 boutons :

- Bouton 1 → Déplacer le curseur à gauche
- Bouton 2 → Déplacer le curseur à droite
- Bouton 3 → Valide le choix de l'utilisateur

Les différents tests effectués sur les bouton pour 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

### **5.2.3 : Scénario pour la commande d'axe :**

- Branchement de la commande d'axe et modification de sa position avec les boutons, puis quand on la rebranche on s'assure qu'elle aille bien à sa nouvelle position.
- Retour à sa position si on la bouge manuellement

## **6 : Conclusion :**

Le but de ce projet était de réaliser le pilotage d'une commande d'axe, pour cela une application a été réalisée sur une carte STM32F411RE avec l'os FreeRTOS, et des tâches ont été mise en place afin de récupérer et modifier la position de la commande d'axe. Il fallait aussi avoir un affichage de la position sur un écran LCD et une vérification de la position. Pour cela différentes tâches ont été créées et mises en place sur la carte. De plus 3 boutons poussoir ont été ajoutés afin d'avoir une interaction avec la commande d'axe. Le pilotage de la commande d'axe a donc été fait. En plus de celle-ci, des fonctions ont été ajoutées afin de reproduire le comportement d'une imprimante. Donc en plus de pouvoir piloter la commande d'axe, l'utilisateur peut aussi imprimer des dessins. Un logiciel en C++ a aussi été fait afin de pouvoir imprimer le dessin que l'on souhaite.