

Rapport Projet

StockAuto



Élèves :

Hugo Gabriel PEREIRA DE ALMEIDA
Florian ALZAIX
Antoine TURMEAU-CIBOIS
Antoine VADOT

Enseignants :

Antoine BULARD
Jordan DUFRESNE

6 juin 2025

Table des matières

1	Introduction	3
2	Conception & réflexion du projet StockAuto	3
2.1	Étude de faisabilité	3
2.2	Conception	3
2.2.1	Schéma fonctionnel niveau 1	3
2.2.2	Schéma fonctionnel niveau 2	4
2.2.3	Matériel nécessaire	4
2.2.4	Conception du véhicule <i>StockAuto</i>	5
2.2.5	Diagramme de séquence	6
3	Alimentation	7
3.1	Besoins en alimentation	7
3.2	Solutions	8
4	Pince	8
4.1	L'idée de la pince	8
4.2	Conception de la pince	9
4.2.1	Conception mécanique	9
4.2.2	Conception électronique	9
4.2.3	Implementation informatique	10
4.3	Vidéo demonstrative	12
5	Moteur pas à pas & Drivers	12
5.1	Choix des moteurs et des drivers	12
5.1.1	Cahier des charges	13
5.1.2	Comparaison des différents moteurs	13
5.1.3	Choix de moteur pas à pas	14
5.2	Mise en place du systèmes des Moteurs	15
5.2.1	Conception mécanique	15
5.2.2	Conception électronique	16
5.2.3	Implementation informatique	18
5.3	Vidéo demonstrative	20
6	Pavé Numérique	20
6.1	L'idée du pavé numérique	20
6.2	Mise en place du pavé	20
6.2.1	Conception électronique	20
6.2.2	Implémentation informatique	22
6.2.3	Configuration des GPIO	22
6.3	Vidéo démonstrative	23

7 Écran OLED	23
7.1 Protocole I2C	23
7.2 Mise en place de l'écran	24
7.2.1 Conception électronique	24
7.2.2 Implémentation informatique	24
7.3 Vidéo démonstrative	26
8 Capteur Optique	26
8.1 Mise en place des capteurs optique	26
8.1.1 Conception électronique	26
8.2 Vidéo démonstrative	27
9 Problèmes majeurs	27
9.1 Pourquoi pas de capteur optique ?	27
9.2 Quels autres problèmes ?	27
10 Boucle principal de notre projet	27
11 Altium	29
12 Vidéo finale du projet	30
13 Bibliographie	31

Table des figures

1 Schéma fonctionnelle de niveau 1	3
2 Schéma fonctionnelle de niveau 2	4
3 Design de notre véhicule	5
4 Structure et points de gravité	6
5 Diagramme de séquence	7
6 LM2596 Module d'alimentation DC DC adjustable	8
7 Montage ServoMoteur & suiveur AOP	10
8 Pièces + engrenage 3D de la glissière	15
9 Structure et transmission 3D	16
10 Circuit driverus stepper	16
11 Circuit moteurs Stepper	17
12 Schéma électrique du moteur Nema 17	18
13 Schéma électrique du pavé numérique	21
14 Montage du pavé numérique	21
15 Schéma électrique de l'écran OLED	24
16 Image du montage des capteurs optique	26
17 Machine à état de notre Main	28
18 PCB Altium	29

1 Introduction

Dans le cadre de notre cours de DEEP (Digital Embedded Electronic Project), nous devions concevoir un projet innovant. Étant dans un groupe composé de membres passionnés par la mécanique, nous avons cherché une idée qui pourrait susciter l'intérêt de chacun. C'est ainsi qu'est née l'idée de StockAuto.

StockAuto est une machine mobile conçue pour faciliter le rangement et la récupération de lourdes boîtes dans de grands casiers. Elle est équipée de deux axes de déplacement : un axe vertical et un axe horizontal, lui permettant d'accéder facilement à différents niveaux et emplacements de stockage.

2 Conception & réflexion du projet StockAuto

2.1 Étude de faisabilité

Cependant, au cours de nos réflexions, nous nous sommes aperçus que, compte tenu du temps qui nous était imparti, il serait compliqué de concevoir les deux axes de mobilité de la pince. Nous avons donc décidé de supprimer l'axe permettant la montée de la pince et de conserver uniquement celui permettant d'aller chercher la boîte dans la case et de la retirer.

2.2 Conception

2.2.1 Schéma fonctionnel niveau 1

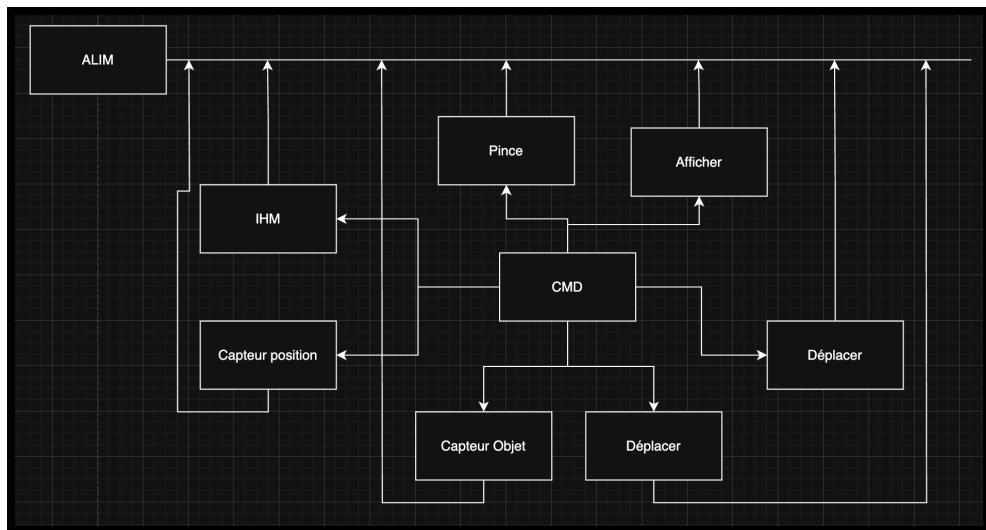


FIGURE 1 – Schéma fonctionnelle de niveau 1

Dans ce schéma fonctionnel (1) nous avons alors défini ce dont nous avions besoin pour notre projet, mais sans trop aller dans les détails. Nous avons les concepts, nous verrons cela dans un second temps dans le schéma fonctionnel de niveau 2 (2) les choses plus en détail

2.2.2 Schéma fonctionnel niveau 2

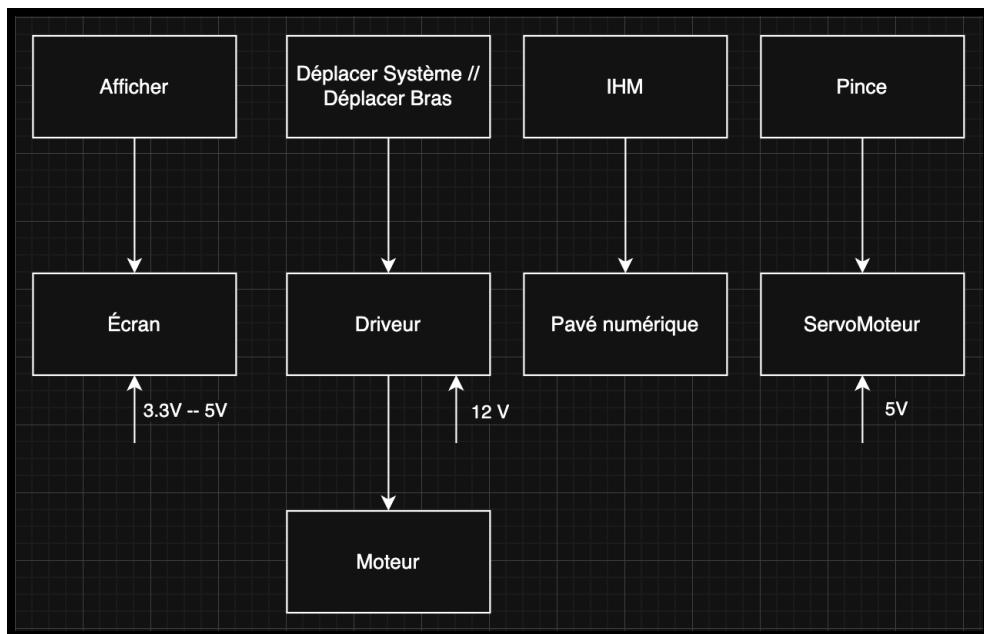


FIGURE 2 – Schéma fonctionnelle de niveau 2

Cependant, dans ce schéma fonctionnel, nous obtenons davantage de précisions sur les composants à utiliser ainsi que sur les alimentations envisageables.

2.2.3 Matériel nécessaire

En prenant en compte notre schéma fonctionnelle cela nous a permis de dresser une liste des composants que nous voulions utiliser au cours de ce projet.

- ×2 Moteur pas à pas (voir comparaison moteurs [5.1.2])
- ×1 Écran Oled
- ×1 Pavé numérique
- ×1 Servomoteur SG90 (voir pourquoi [4.2.1])
- ×2 Capteur Optique

2.2.4 Conception du véhicule *StockAuto*

Une fois le matériel nécessaire au projet établi, nous sommes passés à la phase de conception du design. Nous en avons profité pour calculer les couples nécessaires aux moteurs des roues et de la glissière, afin de pouvoir comparer les différentes options de moteurs, comme nous l'avons fait ici (5.1.2).

Design

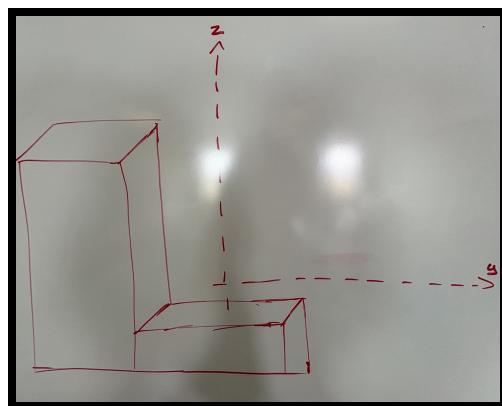


FIGURE 3 – Design de notre véhicule

C'est donc l'idée finale du design que nous voulions mais comme dit précédemment, notre axes sur z nous l'avons supprimé et nous avons garder notre axe y.

Calculs de couple

Nous obtenons alors la masse de nos deux systèmes :

$$P_{\text{1er Système}} = 695 \text{ g} \quad (1)$$

$$P_{\text{2ème Système}} = 829 \text{ g} \quad (2)$$

Nous pouvons alors obtenir les distances à nos centres de gravité, et donc calculer la force nécessaire pour porter notre système :

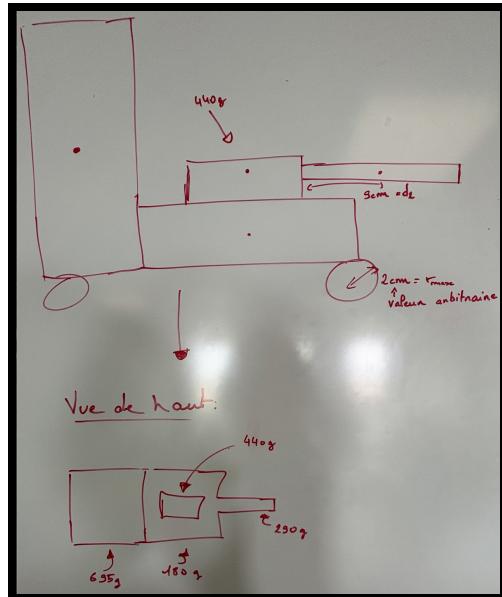


FIGURE 4 – Structure et points de gravité

$$r_{max} = 2 \text{ cm} \quad (3)$$

$$C_1 = m_{sys} \times g \times r_{max} \quad d_2 = 9 \text{ cm} \quad (7)$$

$$C_1 = (829 + 635) \cdot 10^{-3} \times 9.81 \times 2 \cdot 10^{-2} \quad C_1 = m_{sys} \times g \times d_{max} \quad (8)$$

$$C_1 = 290 \cdot 10^{-3} \times 9.81 \times 9 \cdot 10^{-2} \quad (9)$$

$$C_1 = 0.26 \text{ Nm} \quad (10)$$

$$C_1 = 0.29 \text{ Nm} \quad (6)$$

Nous avons donc besoin de deux moteurs qui ont des couples très similaires, en tenant compte du fait que le couple calculé ici est idéal, c'est-à-dire sans frottements ni pertes. Il convient donc de multiplier cette valeur par deux afin d'assurer une marge de sécurité.

2.2.5 Diagramme de séquence

Une fois le concept du projet défini et le matériel nécessaire établi, nous avons réfléchi à la manière dont se déroulerait une routine d'utilisation de notre système. Nous avons donc, à l'aide d'un **schéma de séquence** (5), défini le déroulement pour un utilisateur qui souhaiterait récupérer un objet :

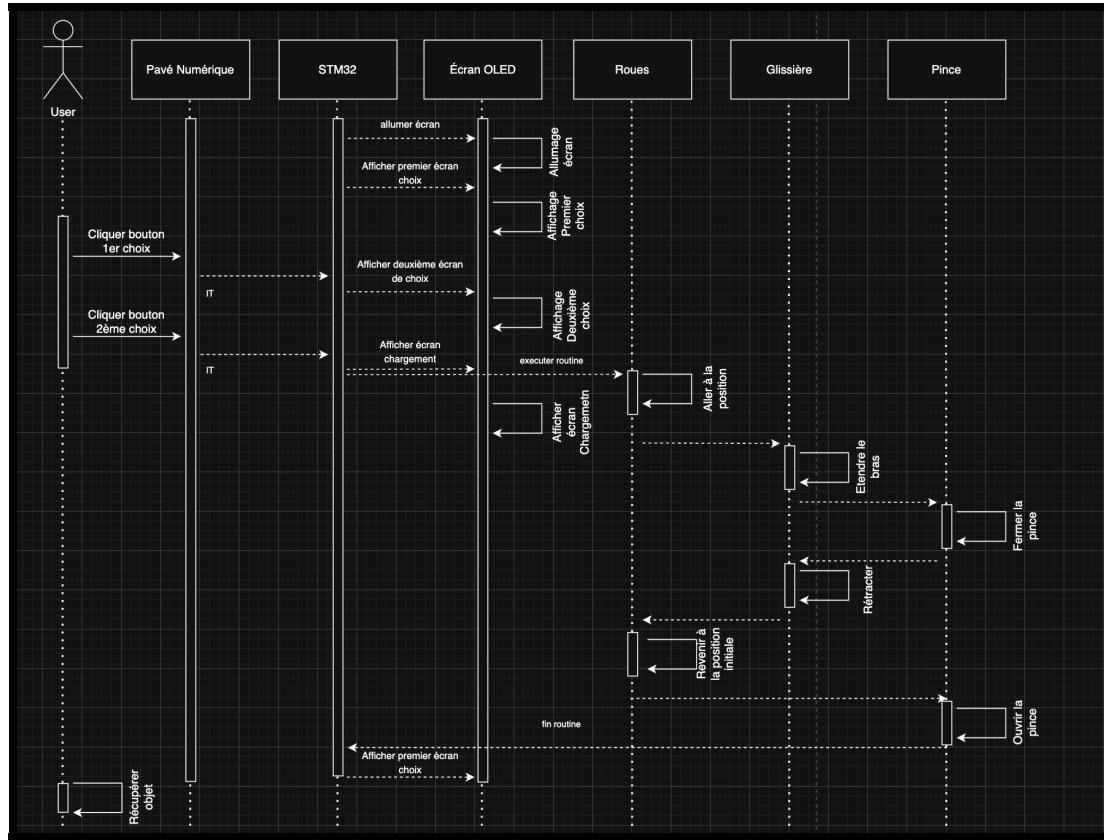


FIGURE 5 – Diagramme de séquence

Nous avons donc notre user qui, en appuyant sur le pavé numérique, lancera une interruption qui permettrait de changer l'affichage de l'écran OLED ainsi que d'exécuter notre routine pour aller chercher un objet.

Cette routine est :

- Aller à la position du casier demandé par l'utilisateur
- Étendre la glissière afin d'approcher la pince de la boîte
- Fermer la pince afin d'attraper la boîte
- Rétracter la glissière
- Revenir à la position de départ
- Ouvrir la pince afin de déposer la boîte

3 Alimentation

3.1 Besoins en alimentation

Comme nous avons pu le voir dans notre schéma fonctionnel de niveau 2 (2), nous allons avoir besoin de deux alimentations différentes. Une en 12 V en ce qui concerne les moteurs

pas à pas et une en 5 V . Nous avons dû alors réfléchir à comment nous allions faire.

3.2 Solutions

En ce qui concerne l'alimentation 12 V nous allons utiliser l'alimentation que nous avons à disposition dans les laboratoires, mais si nous voulions l'utiliser sans nécessité de câble extérieur, il aurait fallu chercher une batterie qui délivre une bonne tension ainsi qu'une bonne intensité.

Cependant, en ce qui concerne l'alimentation 5 V nous n'allons pas réutiliser une deuxième alimentation de labo. Pour cela, nous avons cherché une alimentation à découpage buck :

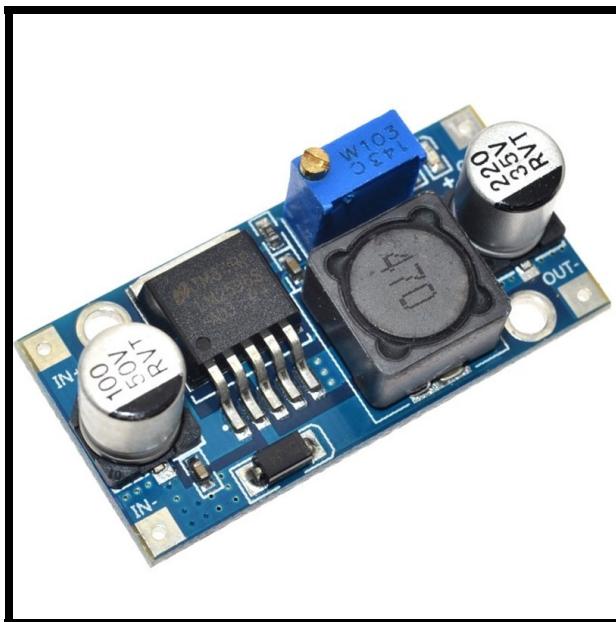


FIGURE 6 – LM2596 Module d'alimentation DC DC ajustable

Nous avons consulté sa documentation [1] et constaté qu'il pouvait accepter jusqu'à 40 V en entrée, et fournir une tension de sortie réglable de $1,5\text{ V}$ à 35 V , grâce au potentiomètre situé sur la boîte bleue.

4 Pince

4.1 L'idée de la pince

Pour réaliser cette pince, nous nous sommes tout d'abord penchés sur les fonctionnalités qu'elle devait offrir, ainsi que sur la méthode à suivre pour la concevoir et la piloter.

Fonctionnalités attendues :

- Capacité à saisir des boîtes de manière stable ;
- Ouverture et fermeture contrôlées par un servomoteur ;
- Robustesse mécanique pour résister au poids des boîtes ;
- Capacité à détecter la prise effective d'une caisse.

4.2 Conception de la pince

4.2.1 Conception mécanique

Pour le modèle de notre pince, nous avons recherché sur Internet des conceptions 3D déjà existantes compatibles avec un servomoteur SG90. Nous avons ainsi trouvé un modèle [2] adapté que nous avons décidé d'utiliser et d'intégrer à notre système. Nous l'avons imprimé en 3D et, pendant ce temps, nous avons commencé à développer le code permettant de piloter le servomoteur.

4.2.2 Conception électronique

Avant même d'entamer le développement du code pour le servomoteur, nous avons réfléchi à la manière de détecter si la pince avait effectivement réussi à saisir un objet. Plusieurs options s'offraient à nous :

- Utilisation d'un capteur de fin de course (endstop) ;
- Capteur de pression piézorésistif ;
- Capteur de pression basé sur une résistance shunt.

Nous nous sommes alors décidés à nous lancer sur la troisième solution, le capteur de pression avec une **résistance shunt**.

Schéma du montage :

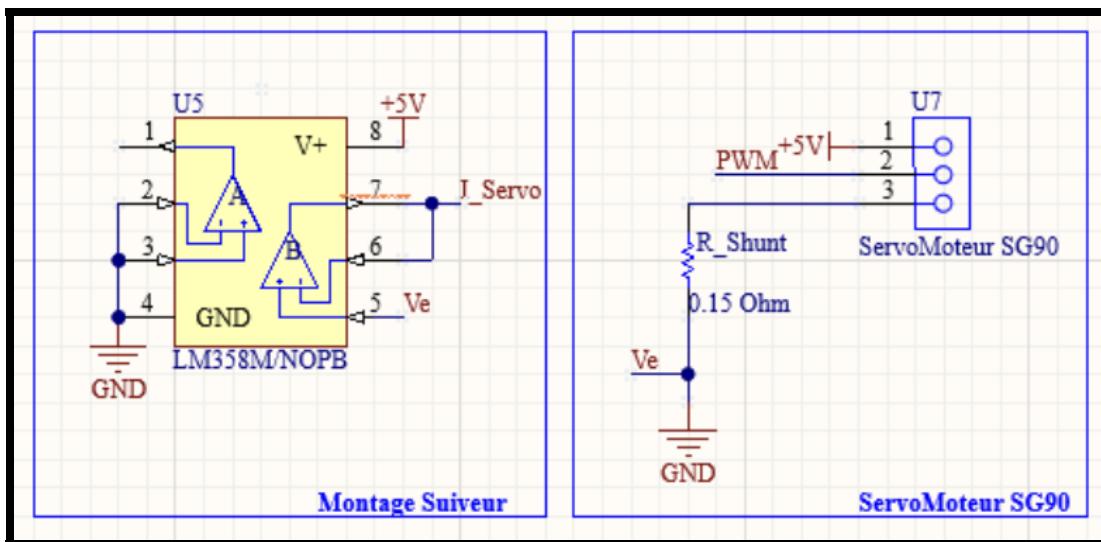


FIGURE 7 – Montage ServoMoteur & suiveur AOP

Nous avons ci-dessus la figure 7 représentant le montage utilisé pour piloter la pince (voir datasheet : [3]). Le servomoteur est alimenté en 5V via notre alimentation buck. Un signal PWM est utilisé pour définir la position du servomoteur. Enfin, une résistance shunt est placée en série entre la masse de l'alimentation et celle du servomoteur, afin de mesurer le courant consommé par le servomoteur, et ainsi déterminer s'il exerce une pression.

Calculs théorique

Nous avons dû calculer à la main, avant de calibrer ensuite, une fois le code et le montage terminés, la valeur potentielle aux bornes de la résistance shunt. Pour faire cela, nous sommes passés par la théorie. Nous savions, grâce à notre générateur, que lorsque le servomoteur se bloquait pendant sa fermeture, il tirait 0,55A. Nous connaissons aussi la valeur de la résistance, mesurée avec notre professeur Bulard Antoine, qui est de 0,15Ω. Nous pouvons alors utiliser la loi d'Ohm.

$$U = R \times I$$

$$U = 0.15 \times 0.55 = 0.08 V$$

Cependant, nous avons par la suite mesuré la tension aux bornes de la résistance, et celle-ci était de 10 mV (soit 0,01 V).

4.2.3 Implementation informatique

Code PWM

En termes de code, nous devions tout d'abord mettre en place un code qui permettrait de piloter notre servomoteur avec un **PWM** [1] :

Code 1 : Définition de la fonction pour initialiser le Servomoteur

```
void INIT_servo(void) {
    BSP_TIMER_run_us(TIMER1_ID, PERIOD * 1000, 0);
    BSP_TIMER_enable_PWM(TIMER1_ID, TIM_CHANNEL_1, DEFAULT_POSITION, 0,
    ↵  0);
}
```

c

Dans cette fonction, nous initialisons avec `BSP_TIMER_run_us()` ; un timer sur le canal souhaité, ici **Timer 1**, et nous définissons une période de 10×1000 , soit **10 ms**.

Le dernier paramètre permet d'autoriser ou non les interruptions ; dans notre cas, celles-ci ne sont pas autorisées.

Avec l'appel de la fonction `BSP_TIMER_enable_PWM()` ;, nous activons le signal PWM sur le canal du timer d'identifiant 1. Le paramètre `DEFAULT_POSITION` correspond à la **duty cycle** que nous souhaitons appliquer à notre PWM. Cette dernière définit combien de temps le signal PWM reste à l'état haut. Cela définira donc la position de notre servomoteur.

Par la suite, nous avons défini des fonctions qui augmentent progressivement le **duty cycle** afin de déplacer le servomoteur dans les deux sens de manière contrôlée.

Code pour capteur pression

Comme mentionné précédemment, la sortie de l'AOP est connectée à la broche PF1. Nous avons donc mis en place les fonctions nécessaires pour récupérer ces valeurs afin de les traiter numériquement.

Code 2 : Définition de la fonction pour lire le PIN adc + filtre numériques

```
uint32_t BSP_ADC_getValue(adc_id_e channel)
{
    if(adc_id[channel] == -1 || channel >= ADC_CHANNEL_NB)
    {
        printf("You asked for the reading of the channel %d which
        ↵  is non initialized or unused! Please review your
        ↵  software\n", channel);
        return -1;
    }
    return adc_converted_value[adc_id[channel]];
}
```

c

```

}

uint32_t Read_ADC_Averaged(uint8_t num_samples) {
    uint32_t sum = 0;
    for (uint8_t i = 0; i < num_samples; i++) {
        sum += BSP_ADC_getValue(ADC_1);
        HAL_Delay(1); // Petite pause entre les lectures
    }
    return sum / num_samples;
}

```

Notre première fonction `BSP_ADC_getValue()`, en lui donnant l'ID de la pin ADC que nous utilisons, permet de lire la valeur de cette dernière. De plus, si elle remarque que la pin donnée n'est pas configurée en tant qu'ADC, elle affichera un message pour nous en informer.

Pour la deuxième, `Read_ADC_Averaged()`, c'est une fonction qui permet de prendre '`num_samples`' valeurs pour en faire une moyenne. C'est donc un filtrage numérique, et plus on augmenteras la valeur de `num_samples` plus notre valeur finale sera filtré.

Nous avons donc défini un seuil afin de pouvoir arrêter la pince, en lui appliquant un **Duty Cycle** fixe une fois que cette dernière a saisi l'objet tout cela avec ce bout de code :

Code 3 : Définition du seuil pour arrêter la pince

```

if (voltage_mv > 10) {
    hold_(step_position);
    break;
}

```

Ici donc si ma valeur que je lis et filtre dépasse les 10mV alors on **hold** la position

4.3 Vidéo demonstrative

Afin de pourvoir voir la pince seule en action vous pouvez aller sur cette vidéo non répertorié :

[Cliquer ici pour voir la vidéo](#)

5 Moteur pas à pas & Drivers

5.1 Choix des moteurs et des drivers

Afin de sélectionner les moteurs et les drivers adaptés à notre projet, nous avons tout d'abord établi un cahier des charges. Celui-ci nous a permis de comparer les différentes

options disponibles et de choisir les composants répondant le mieux à nos besoins.

5.1.1 Cahier des charges

Les moteurs devaient répondre aux critères suivants :

- Fournir un couple suffisant pour supporter l'ensemble de la masse du projet. (Couple nécessaires à voir ici (2.2.4))
- Offrir une grande précision de déplacement, afin de s'arrêter précisément devant les étagères.
- Pour pouvoir revenir à la position de base, il est nécessaire de connaître le déplacement total effectué vers une position donnée.

5.1.2 Comparaison des différents moteurs

Afin de sélectionner le moteur le plus adapté à notre application, nous avons comparé plusieurs modèles en nous basant sur les critères définis dans notre cahier des charges

Moteur Brushless

Le moteur brushless est un moteur électrique qui utilise des aimants permanents pour créer un champ magnétique entraînant le mouvement du rotor. Contrairement aux moteurs à balais, la commutation est réalisée électroniquement, ce qui améliore les performances et la durabilité du moteur.

Avantages :

- Très bon rendement énergétique.
- Faible échauffement.

Inconvénients :

- Nécessite un contrôleur électronique spécifique (ESC : Electronic Speed Controller).
- Pas adapté aux applications nécessitant un positionnement angulaire précis.

Donc en ce qui concerne ce type de moteur il n'est pas adapté à nos besoins

Moteur courant continue

Le moteur à courant continu ou DC est un moteur électrique pilotable en envoyant une tension à ces bornes, ça crée un champ électrique ce qui entraîne le rotor.

Avantages :

- Commande simple
- Bonne réactivité en vitesse.

Inconvénients :

- Moins précis pour le positionnement sans capteurs.

Encore une fois, ce moteur ne correspondait pas à ce que nous recherchions. En effet, la vitesse n'était pas une caractéristique que nous voulions ; nous avions besoin, avant tout, de précision et de couple.

Moteur pas à pas

Le moteur pas à pas est un moteur électrique qui permet de transformer une impulsion électrique en un mouvement angulaire

Avantages :

- Très grande précision de positionnement.
- Facile à contrôler en boucle ouverte.
- Bon couple à basse vitesse.
- Idéal pour les mouvements répétitifs et contrôlés.

Inconvénients :

- Moins efficace à haute vitesse.
- Peut chauffer si mal configuré.
- Nécessite un driver spécifique pour fonctionner.

Le moteur pas à pas est donc, contrairement aux autres, celui qui correspond le mieux à nos besoins. Il est précis, car en envoyant un certain nombre d'impulsions, on peut le faire tourner d'un angle défini. De plus, il offre un bon couple à basse vitesse. Or, notre projet ne nécessite pas de grande vitesse, c'est pourquoi nous avons choisi ce type de moteur.

5.1.3 Choix de moteur pas à pas

En faisant quelques recherche sur internet nous nous sommes arrêter sur le modele Nema 17 [4]

Spécifications

- **Dimensions :** $42 \times 42 \times 39mm$
- **Couple :** $42Ncm$
- **Courant Nominal :** $1.5A$
- **Poids :** $0.26Kg$

Nous avons constaté un couple de $42 N\cdot cm$, ce qui est suffisant en comparaison avec les calculs que nous avions effectués auparavant pour faire rouler notre système.

Cependant, pour contrôler ce moteur, nous avions besoin de driveurs. Cela n'a pas été très difficile à trouver, car il existe des driveurs spécialement conçus pour ce type de moteur. Nous avons alors trouver c'est module de Pilotage de Moteur AZDelivery A4988. [5]

5.2 Mise en place du systèmes des Moteurs

5.2.1 Conception mécanique

Avant même d'entamer le développement du code ou le montage électronique des moteurs, nous avons dû réfléchir à la conception de l'ensemble des éléments mécaniques associés. Cela inclut notamment la mise en place d'une glissière et d'un système de transmission permettant de faire tourner deux de nos quatre roues.

Glissière

Notre objectif était de contrôler une glissière à l'aide de l'un de nos moteurs. Cette glissière était initialement prévue pour être commandée, cependant, elle n'est jamais arrivée. Nous avons donc décidé de la concevoir nous-mêmes en 3D, afin de pouvoir l'imprimer et l'intégrer à notre système.

Vous pouvez voir le design ci-dessous (8) :



FIGURE 8 – Pièces + engrenage 3D de la glissière

Transmission

En ce qui concerne les roues de notre véhicule, nous ne disposons que d'un seul moteur pour en piloter deux roues. Nous avons donc conçu toute une structure permettant de fixer le moteur et d'assurer la transmission du mouvement aux deux roues.

Pour cela, nous avons utilisé des pièces de Lego afin de créer un système de transmission simple et fonctionnel, tout en garantissant un bon alignement et stabilité grâce à la structure.

Vous pouvez voir le design ci-dessous (9) :

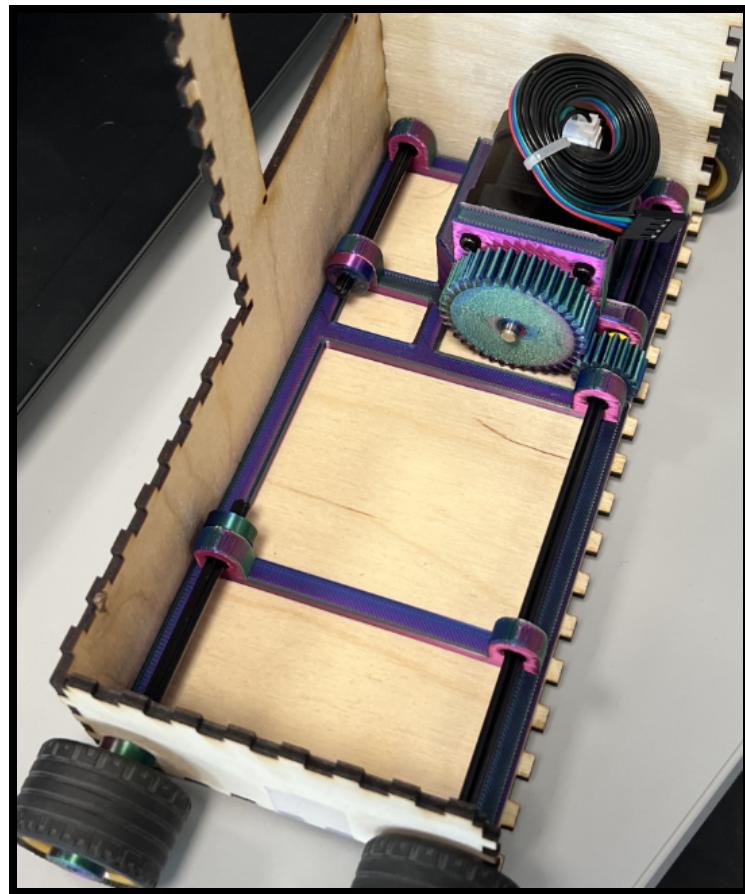


FIGURE 9 – Structure et transmission 3D

5.2.2 Conception électronique

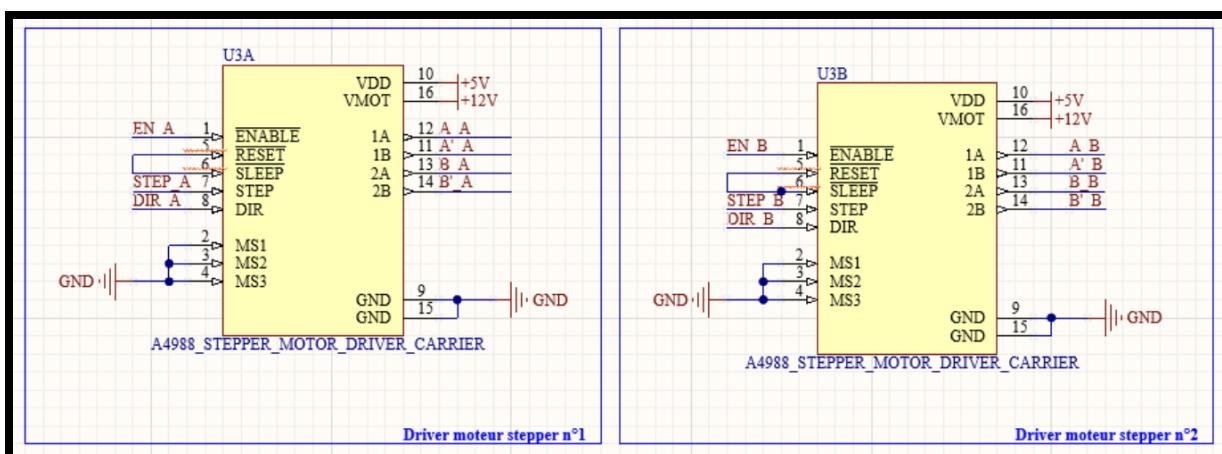


FIGURE 10 – Circuit driverus stepper

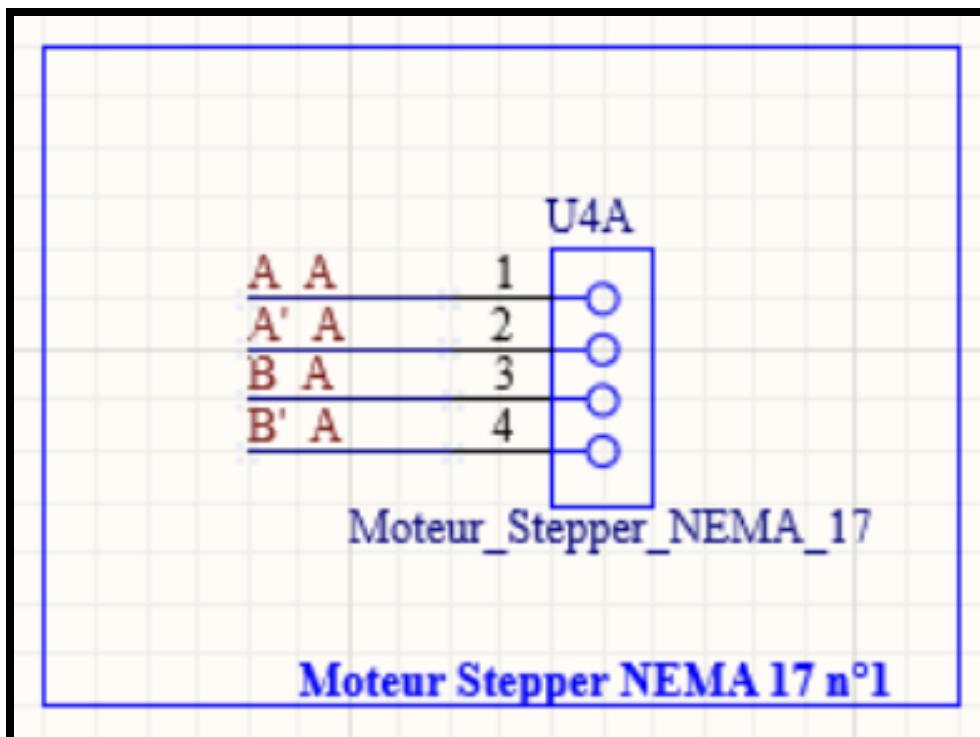


FIGURE 11 – Circuit moteurs Stepper

Sur les montages ci-dessus, nous avons nos deux driveurs ainsi que un de nos moteur Nema 17. Concernant le driveur, nous avons plusieurs pin et leur fonctionnement qu'on à trouver dans leurs datasheet [6] :

- **ENABLE** : Il sert à alumer/éteindre le moteur.
- **RESET** : Il sert à réinitialiser les registres internes au moteur (Nous avons décidé de le relier au SLEEP).
- **SLEEP** : Il sert à "éteindre" le moteur (Nous avons décidé de le relier au RESET).
- **STEP** : C'est la pin qui va permettre de définir quand le moteur fait un pas.
- **DIR** : Cela nous permet de définir la direction de notre moteur.
- **MS1/MS2/MS3** : Ces trois pin nous permettent de définir la taille d'un pas. 1/6 de pas ou 1/12 de pas ou 1/16 de pas ou pour finir 1/32 pas. Ici nous sommes en 1/16 de pas.
- **VDD** : Tension 5V envoyé par la carte .
- **VMOT** : Tension 12V d'entrée.
- **GND** : Un qui vient de la STM et l'autre de l'alimentation.
- **A - A' - B - B'** : Ce sont les pins qui seront reliés au moteur Stepper

Concernant le moteur maintenant nous n'avons que 4 pins

- **A - A - A - A** : Pins qui permetent de piloter notre moteur. (En réalité les pins sont A - C - B - D)

Afin de savoir à quoi correspond chaque broche nous avons du nous pencher un peu plus sur ce qui se passe à l'intérieur de notre moteur (12) :

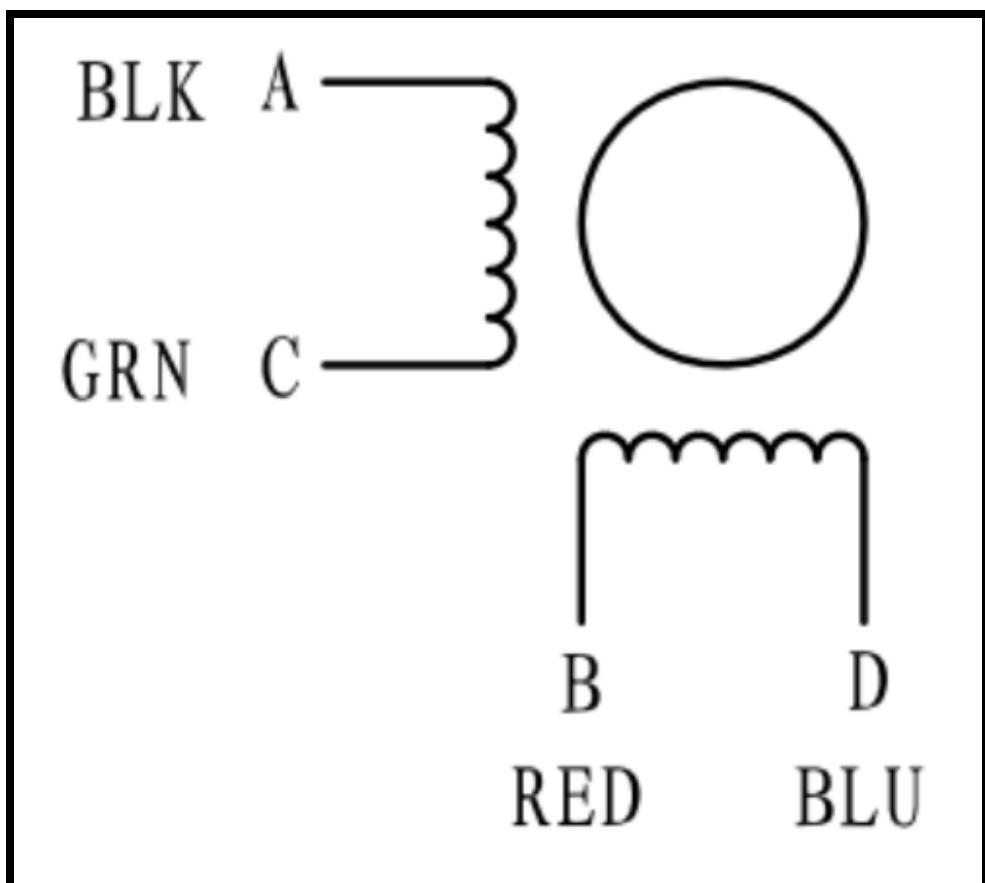


FIGURE 12 – Schéma électrique du moteur Nema 17

Nous avons dans ce moteur deux enroulements. Chaque enroulement est une inductance (bobine). En envoyant un courant entre A et C, par exemple, cela va créer un champ magnétique qui va faire bouger le rotor et donc déplacer le moteur d'un pas. Pour piloter ce moteur, il nous faut contrôler ces deux bobines alternativement. C'est donc le travail de nos driveurs : une fois que ce dernier reçoit un signal PWM, il va alternativement envoyer un courant sur chaque bobine. Suite à cela il à fallut passer au code qui permet d'envoyer les instruction pour le moteur.

5.2.3 Implementation informatique

Code Driveurs

Afin de pourvoir piloter les driveurs et donc les moteurs, nous avons du choisir des pins de notre carte qui marcherais en tant que GPIO. Tout cela nous l'avons réalisée sur STMCubeIDE ce qui à pu nous donner tout un fichier de config. Cependant nous allons

nous pencher sur une de nos configuration. Pour le step nous avons utilisé les GPIO comme un PWM.

Code 4 : Code pour la broche Step des Driveurs

```
for (int i = 0; i < number_step; i++) {
    HAL_GPIO_WritePin(GPIOA, PIN_STEP_A, GPIO_PIN_SET);      // STEP HIGH
    HAL_Delay(1);
    HAL_GPIO_WritePin(GPIOA, PIN_STEP_A, GPIO_PIN_RESET);   // STEP LOW
    HAL_Delay(1);
}
```

Mais pourquoi ne pas avoir choisi d'utiliser directement un PWM ?

La raison qui nous a fait choisir cette option plus qu'un PWM est qu'avec cette solution, ce que nous mettons dans notre boucle `for()` c'est l'exact nombre de pas que nous allons effectuer, ce qui peut alors nous permettre de simplement définir les pas nécessaires pour chaque casier et de nous simplifier la tâche. De plus, malgré que ce soit des lignes bloquantes au niveau du reste du code, rien d'autre ne doit réellement se passer pendant que notre véhicule ou que notre glissière se déplace, ce n'est donc pas dérangeant.

De plus car nous avons 2 driveurs il a fallut gérer chaque driveurs indépendamment l'un de l'autre, en effet si un moteur n'est pas utiliser nous souhaitons pouvoir l'éteindre pendant que l'autre lui est allumé.

Code 5 : GPIO_step

```
void GPIO_step(int number_step, int direction, char driveur) {
    switch (driveur) {
        case 'A':
            HAL_GPIO_WritePin(GPIOA, PIN_EN_A, GPIO_PIN_RESET);

            if (direction == 1) {
                HAL_GPIO_WritePin(GPIOA, PIN_DIR_A, GPIO_PIN_SET);
            } else {
                HAL_GPIO_WritePin(GPIOA, PIN_DIR_A, GPIO_PIN_RESET);
            }

            for (int i = 0; i < number_step; i++) {
                HAL_GPIO_WritePin(GPIOA, PIN_STEP_A, GPIO_PIN_SET);
                HAL_Delay(1);
                HAL_GPIO_WritePin(GPIOA, PIN_STEP_A, GPIO_PIN_RESET);
                HAL_Delay(1);
            }

            HAL_GPIO_WritePin(GPIOA, PIN_EN_A, GPIO_PIN_SET);
    }
}
```

```
        break;

    case 'B':
        //Même logique mais avec le le driveur B
    default:
        Error_Handler();
        break;
    }
}
```

Avec ce code (code : 5), nous pouvons alors piloter le driver et donc le moteur qu'on souhaite faire tourner, tout en donnant une direction ainsi qu'un nombre de steps.

5.3 Vidéo demonstrative

Afin de pourvoir voir un moteur seul en action vous pouvez aller sur cette vidéo non répertorié :

[Cliquer ici pour voir la vidéo](#)

6 Pavé Numérique

6.1 L'idée du pavé numérique

Le pavé numérique à pour objectif de pouvoir permettre à l'utilisateur d'interagir avec le véhicule, c'est avec ce dernier que nous choisirons si nous voulons prendre ou déposer des objets dans les casiers et dans un second temps cela nous permettras de choisir un numéro de casier.

6.2 Mise en place du pavé

6.2.1 Conception électronique

Comment fonctionne le pavé

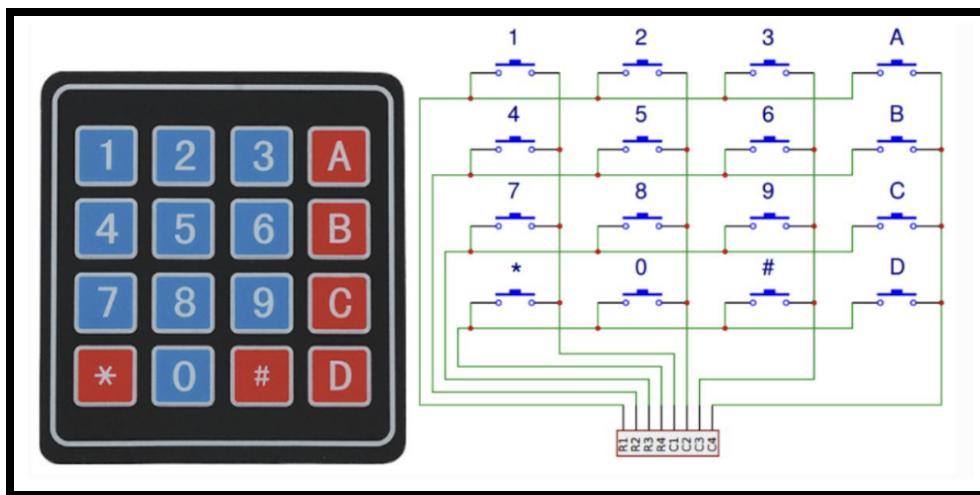


FIGURE 13 – Schéma électrique du pavé numérique

Juste au-dessus (13), on voit que le clavier matriciel 4x4 fonctionne comme une grille composée de 16 interrupteurs (voir datasheet : ??), organisés en lignes et colonnes. Chaque touche relie une ligne à une colonne : lorsqu'on appuie dessus, cela établit un contact électrique entre les deux.

Pour savoir quelle touche a été pressée, le microcontrôleur balaye les lignes une par une, en envoyant un signal. En parallèle, les colonnes sont placées en mode “écoute” (elles sont maintenues à l'état haut). Si une touche est appuyée, la colonne reliée à la ligne activée passe à l'état bas. Le système peut alors facilement identifier la ligne et la colonne concernées, et donc savoir exactement quelle touche a été enfoncee.

Montage

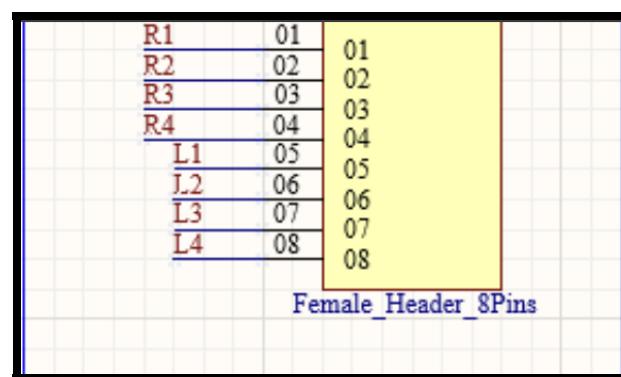


FIGURE 14 – Montage du pavé numérique

Comme vu ici (14), nous avons donc 8 pins à brancher. Ces huit pins sont des GPIO. 4 de ces pins sont là afin d'obtenir le signal de la ligne du bouton pressé et les 4 autres pour la colonne.

6.2.2 Implémentation informatique

Dans notre projet, pour détecter quelle touche du clavier matriciel est pressée, nous activons les lignes une par une. Concrètement, cela consiste à mettre la ligne que l'on souhaite tester à l'état bas (LOW), tandis que les autres restent à l'état haut (HIGH). Ensuite, nous lisons l'état des colonnes. Si l'une d'elles passe à l'état bas, cela signifie qu'une touche a été pressée à l'intersection de la ligne active et de cette colonne. En répétant cette opération pour chaque ligne, on peut identifier précisément la touche appuyée. Une table de correspondance (keymap) nous permet alors d'associer chaque combinaison ligne/colonne à une touche donnée.

Par exemple, si la ligne 2 est à LOW et que la colonne 3 détecte un état LOW également, on en déduit que la touche située à l'intersection (ligne 2, colonne 3) est pressée.

6.2.3 Configuration des GPIO

En ce qui concerne les GPIO du pavé numérique, nous les avons configurés de manière à ce que l'appui sur une touche déclenche une interruption. Cela permet de lever un flag d'interruption et de traiter immédiatement la donnée envoyée par le clavier grâce à une routine d'interruption (code : 6).

Code 6 : Routine d'interruption GPIO

```

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    for (int row = 0; row < 4; row++) {
        for (int i = 0; i < 4; i++)

            HAL_GPIO_WritePin(rowPorts[row], rowPins[row], GPIO_PIN_RESET);

        for (int col = 0; col < 4; col++) {
            if (colPins[col] == GPIO_Pin) {
                if (HAL_GPIO_ReadPin(colPorts[col], colPins[col]) ==
                    GPIO_PIN_RESET) {
                    char touche = keymap[row][col];

                    pressed_flag = 1;

                    last_key = touche;
                }
            }
        }
    }
    for (int i = 0; i < 4; i++)
        HAL_GPIO_WritePin(rowPorts[i], rowPins[i], GPIO_PIN_RESET);
}

```

c

Par la suite, nous avons simplement créé une fonction (code : 7) permettant de récupérer la touche pressée. Cette fonction est bloquante, ce qui signifie que tant qu'aucune touche n'est appuyée sur le clavier, le programme reste en attente et ne poursuit pas son exécution. Ce dernier attend une interruption afin de poursuivre.

Code 7 : Fonction pour récupérer la touche

```
char keypad_getkey(void) {
    while(!pressed_flag) HAL_Delay(25);

    pressed_flag = 0;
    HAL_Delay(50);

    return last_key;
}
```

c

6.3 Vidéo démonstrative

Afin de pourvoir voir le pavé seul en action vous pouvez aller sur cette vidéo non répertorié :

[Cliquer ici pour voir la vidéo](#)

7 Écran OLED

7.1 Protocole I2C

Le protocole I2C est un bus de communication série qui se base sur deux fils :

- SCL (Serial Clock Line) : ligne d'horloge, qui synchronise les échanges.
- SDA (Serial Data Line) : ligne de données, par laquelle transittent les informations.

Cette communication fonctionne selon une architecture maître-esclave :

- Le maître : c'est lui qui contrôle le bus. Il génère le signal d'horloge et initie les communications avec les esclaves.
- L'esclave : Ce dernier répond aux requêtes. Chaque esclave posséde une adresse unique.

Et c'est ce dernier qui rend l'I2C intéressant, parce que chaque composant qui communique en I2C a une adresse différente, ce qui veut dire que, sur les mêmes deux fils, nous pouvons communiquer à plusieurs composants en mettant simplement l'adresse de ce dernier.

7.2 Mise en place de l'écran

7.2.1 Conception électronique

En regardant donc la documentation de notre écran OLED [7], nous avons vu que c'était une communication I2C et que donc nous aurions besoin que de 3 câbles. Un qui servirait pour l'alimentation, et les deux autres le SCL et le SDA afin d'établir la connexion I2C.

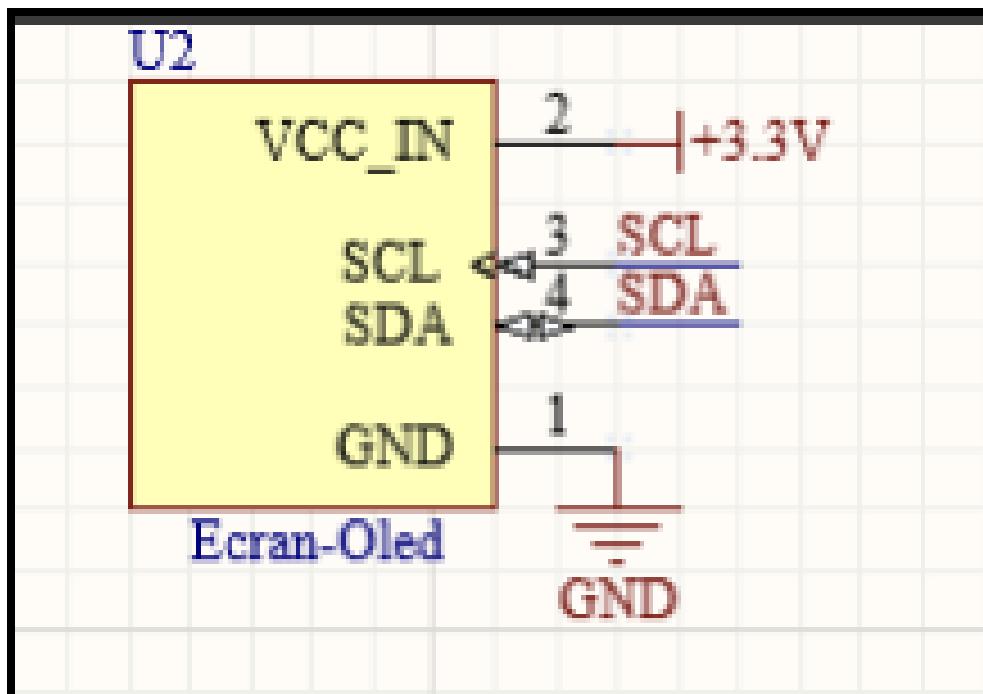


FIGURE 15 – Schéma électrique de l'écran OLED

Comme vu au dessus (15) notre écran est alimenté en 3.3 V, cette tension nous sera fournie par la STM32.

7.2.2 Implémentation informatique

Implémentation classe I2C

Afin de nous simplifier la tâche de la communication I2C, nous avons fait des fonctions qu'on pourrait ensuite utiliser :

Code 8 : Fonction Send Command

```
void OLED_SendCommand(uint8_t command)
{
    uint8_t data[2] = {0x00, command};
    I2C_TransmitDualByte(OLED_ADDR, data);
```

c

}

Nous avons consulté la documentation [7], et après une analyse approfondie, nous avons compris qu'avant d'envoyer n'importe quelle commande à l'écran, il est nécessaire de transmettre en amont l'octet 0x00. Cela permet à l'écran de savoir que les données qui suivront correspondent à une commande. (voir code : 8)

Code 9 : Fonction Send Data

```
void OLED_SendData(uint8_t *data, uint16_t dataSize)
{
    uint8_t buffer[dataSize + 1];
    buffer[0] = 0x40;
    memcpy(&buffer[1], data, dataSize);
    I2C_TransmitBytes(OLED_ADDR, buffer, sizeof(buffer));
}
```

A contrario, quand nous voulons envoyer de la data, ce n'est plus 0x00 mais bien 0x40 (voir code : 9)

Code 10 : Séquence d'initialisation

```
uint8_t init[] = {
    0xA8, 0x3F,
    0xD3, 0x00,
    0x20, 0x00,
    0x40,
    0xA1,
    0xC8,
    0xDA, 0x12,
    0x81, 0x7F,
    0xA4,
    0xA6,
    0xD5, 0x80,
    0x8D, 0x14,
    0xAF};
```

Cependant, avant toute chose, avant même d'envoyer des données, il est nécessaire d'initialiser l'écran. Cette procédure d'initialisation est décrite dans la documentation [7]. Une fois l'écran correctement initialisé à l'aide de la fonction `OLED_Init`, il devient alors possible d'envoyer des données.

7.3 Vidéo démonstrative

Afin de pourvoir voir l'écran en action vous pouvez aller sur ces vidéos non répertorié :

[Cliquer ici pour voir cette première vidéo](#)

[Cliquer ici pour voir cette deuxième vidéo](#)

8 Capteur Optique

Nous avions comme intention d'utiliser ces capteurs afin de calibrer notre véhicule. Car si, par exemple, notre véhicule s'arrête en pleine routine et que donc il n'est pas à sa position initiale, il y aurait au début une fonction de calibration qui regarderait si le capteur est enclenché ou pas et si ce n'est pas le cas, déplacerait la voiture afin de calibrer. Cependant vous verrez ici (9.1) que nous n'avons pas pu l'utiliser.

8.1 Mise en place des capteurs optique

8.1.1 Conception électronique

Nous avons trouvé alors la datasheet de nos capteurs qui se trouve ici : [8]

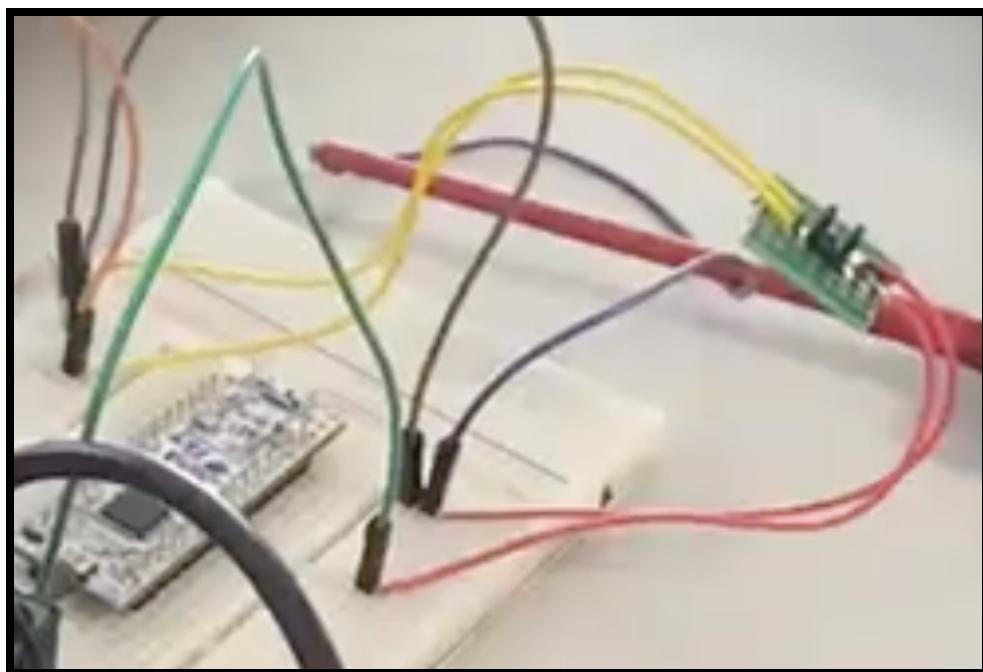


FIGURE 16 – Image du montage des capteurs optique

Le capteur **EE-SX1330** possède deux parties principales à câbler :

- **Côté LED (émetteur)** : La LED infrarouge doit être alimentée entre les broches **anode** et **cathode**. Pour cela, on connecte :
 - L'anode à la tension d'alimentation (+5 V),
 - En insérant en série une résistance de $220\ \Omega$ pour limiter le courant,
 - Et la cathode à la masse (GND).
- **Côté phototransistor (récepteur)** : Le phototransistor possède un **collecteur** et un **émetteur**.
 - Le **collecteur** est relié à la tension d'alimentation (+5 V) à travers une résistance de **pull-up** (environ $10\ k\Omega$),
 - Le **collecteur** est aussi connecté à une entrée numérique du microcontrôleur, nous nous sommes simplement arrêtés à lire la valeur dans un voltmètre,
 - L'**émetteur** est connecté à la masse (GND).

8.2 Vidéo démonstrative

Afin de pourvoir voir un capteur en action vous pouvez aller sur cette vidéo non répertorié :

[Cliquer ici pour voir la vidéo](#)

9 Problèmes majeurs

9.1 Pourquoi pas de capteur optique ?

Lors de ce projet nous nous sommes vite confronté à quelques problèmes qui étaient tous liée à la même sources.

Notre problème principale était qu'on utilisais 100% des pins de la carte ce qui à fait que nous n'avons pas pu mettre en place la solution des capteurs optique.

9.2 Quels autres problèmes ?

Dans un second temps, cela nous a impactés, car l'utilisation du I2C nécessitait que certains autres pins soient libres, ce qui a fait que nous avons dû débrancher le pin d'une des lignes de notre pavé afin de pouvoir toujours piloter nos moteurs stepper.

10 Boucle principal de notre projet

Afin de pourvoir structurer le main de notre projet, en reprenant notre diagramme de séquence (5), nous en avons fait une machine à état :

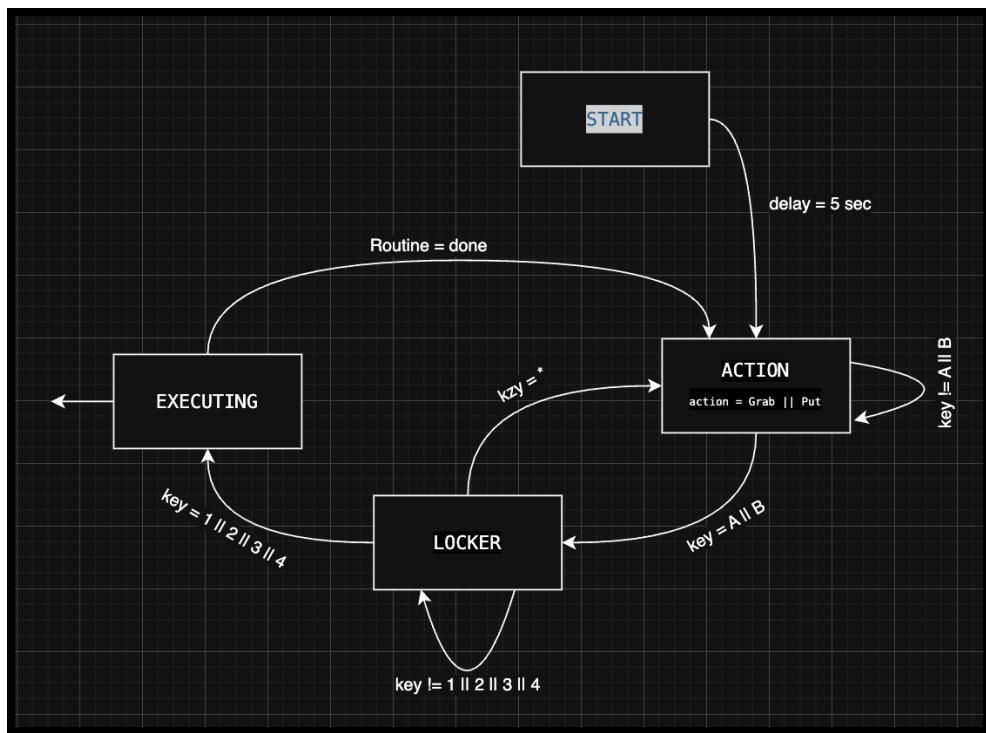


FIGURE 17 – Machine à état de notre Main

Dans notre état *executing*, la flèche partant vers la gauche représente la routine d'exécution, que l'on peut retrouver dans le diagramme de séquence (voir figure 5). C'est à ce moment que la voiture se déplace, exécute les actions planifiées, etc. À la fin de l'exécution on passe alors à notre état *ACTION*.

11 Altium

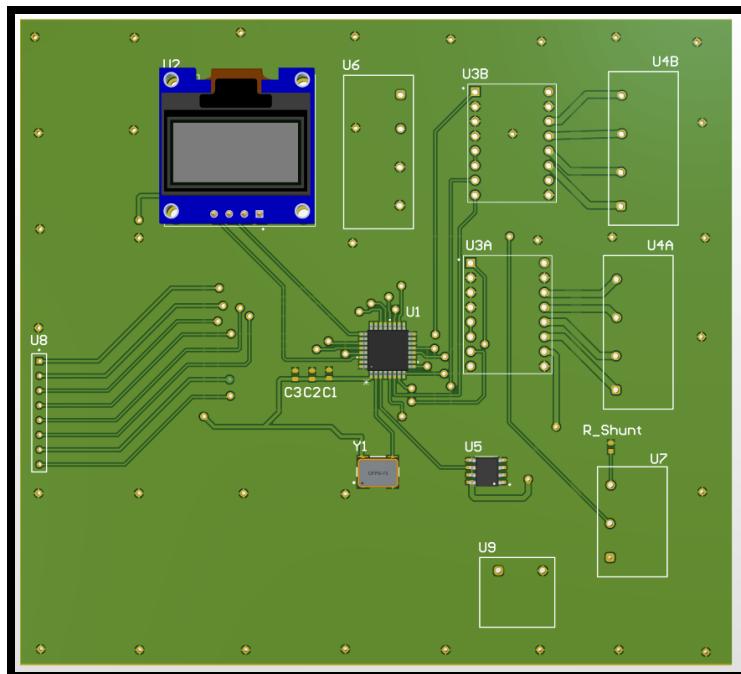


FIGURE 18 – PCB Altium

Dans ce projet, nous avons ensuite réalisé le routage complet de notre carte PCB. Nous avons choisi de respecter les standards IPC Classe 2, largement utilisés dans l'industrie pour les applications générales, embarquées et industrielles. Ce standard impose notamment une largeur minimale de piste de 10 mil et un espacement minimal de 10 mil entre les pistes (10 mil = 0,254 mm). La carte a été conçue avec des dimensions de 125 mm × 125 mm. Par ailleurs, nous avons remplacé les moteurs initiaux par des borniers à vis, ce qui facilite les connexions.

Une barrette femelle 8 broches a également été ajoutée sur le PCB, permettant de connecter facilement notre pavé numérique. Cette solution offre un montage propre, réversible, et compatible avec nos besoins de prototypage. Lors du routage, nous avons naturellement évité les angles droits sur les pistes, afin de réduire les réflexions de signal et d'améliorer la fiabilité électrique du tracé.

Adaptation de la largeur des pistes

Certaines pistes ont vu leur largeur augmentée en fonction du courant qu'elles transportent. Par exemple, la piste alimentant le +12V a été portée à 70 mil, afin de garantir une section suffisante pour éviter tout échauffement ou chute de tension excessive.

De plus, chaque piste traversant une couche interne (Mid Layer 1 ou 2) a été élargie pour compenser la plus faible dissipation thermique et la complexité de fabrication. Une piste

initialement prévue en 10 mil sur la couche externe a ainsi été portée à 15 mil lorsqu'elle passait sur une couche interne, conformément aux bonnes pratiques de fabrication.

Architecture de la carte

Notre PCB est composé de 4 couches :

- Top Layer : utilisée pour les pistes de signal et un plan local de masse.
- Mid Layer 1 : dédiée aux signaux, pour séparer les flux critiques.
- Mid Layer 2 : utilisée pour la distribution du +5V.
- Bottom Layer : utilisée comme plan de masse principal (GND).

Qualité, robustesse et production

Nous avons pris soin de répartir des vias de liaison à la masse tout autour de la carte, notamment autour des plans d'alimentation. Cela permet de garantir une continuité du plan de masse, de réduire l'inductance parasite et de limiter les boucles de courant haute fréquence, ce qui est essentiel pour la stabilité électromagnétique.

De plus, des protections thermiques ont été ajoutées sur les pastilles reliées aux plans de masse. Cela permet de faciliter la soudure manuelle ou automatique de ces pads, en limitant la dissipation thermique directe dans le cuivre environnant.

Enfin, un contrôle DRC (Design Rule Check) a été réalisé pour garantir la conformité aux règles de fabrication. La conception a également intégré l'ajout de condensateurs de découplage au plus près des composants sensibles, ainsi que des points de test pour simplifier la mise au point.

12 Vidéo finale du projet

Nous présentons ci-dessous une vidéo finale de démonstration de notre projet :

[Cliquer ici pour voir la vidéo](#)

13 Bibliographie

- [1] TI. *Datasheet du module LM2596*. URL : <https://www.ti.com/lit/ds/symlink/lm2596.pdf>.
- [2] ALARIK72. *Modèle 3D pince - Robot Gripper 9g Micro Servo*. URL : <https://cults3d.com/fr/mod%C3%A8le-3d/gadget/robot-gripper-9g-micro-servo-for-ar2-or-ar3-robot>.
- [3] FRIENDLYWIRE. *SG90 9 g Micro Servo*. URL : <https://www.friendlywire.com/projects/ne555-servo-safe/SG90-datasheet.pdf>.
- [4] AMAZON. *Lien des moteur Nema 17*. URL : <https://rb.gy/8snkx7>.
- [5] AMAZON. *Lien des driveurs moteur Nema 17*. URL : <https://rb.gy/otuwi1>.
- [6] ALLEGRO MICROSYSTEMS. *A4988 DMOS Microstepping Driver*. URL : https://www.pololu.com/file/0j450/a4988_dmos_microstepping_driver_with_translator.pdf.
- [7] SOLOMON SYSTECH LIMITED. *OLED/PLED Segment/Common Driver with Controller*. URL : <https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf>.
- [8] SOLOMON SYSTECH LIMITED. *Data sheet Capteur Optique*. URL : https://omronfs.omron.com/en_US/ecb/products/pdf/en-ee_sx1330.pdf.