Rapport de projet

Section Mécatronique et Système Complexes (MSC)

Ecole Nationale Supérieure Electronique et ses Applications (ENSEA)

Robobeer

Élèves :

- DANDACHE Ali
- MÉLIN Julien
- PERTUET Leo
- PHILIPPE Quentin
- SCHINI Ludovic

Professeurs encadrants :

- MARTIN Alexis, Responsable Spécialité MSC et responsable adjoint du Département Electronique et Energie. Enseignant ENSEA
- PAPAZOGLOU Nicolas, Responsable Spécialité AEI, Enseignant ENSEA

GitHub: https://github.com/alidanda93/RoboBeer


FEETECH
FS5103B

Remerciements

Pour débuter ce rapport, nous tenons chaleureusement à remercier tous les acteurs qui sont intervenus de prêt ou de loin dans la réalisation de ce projet étudiant. Nous tenons à remercier plus particulièrement :

- Monsieur Nicolas PAPAZOUGLOU et Monsieur Alexis MARTIN, professeurs à l'ENSEA et encadrants de ce projet pour l'apport de leurs expertises multiples que ce soit pour la conception Hardware ou la programmation du système embarqué. Cet accompagnement tout au long des 11 séances nous a permis de mettre en application de nombreux outils vu durant notre cursus à l'ENSEA mais aussi de challenger notre capacité à mener et gérer un projet dans les meilleures conditions.

- Madame Patricia KITTEL pour nous avoir consacré du temps à réaliser nos circuits imprimés mais également pour son approche pédagogique qui invite à la curiosité. Nous avons notamment pu découvrir le processus de réalisation d'un PCB ainsi que celui de dépose de composants CMS au four.

- Monsieur Laurent FIACK, professeur à l'ENSEA à l'initiative des projets robobeer de 3ème année pour avoir tenté de nous déstabiliser afin qu'on ne batte pas ses étudiants qui partaient avec une meilleure main que nous.

- l'ENSEA pour avoir proposé ces projets de dernière année et nous avoir alloué un budget pour pouvoir les réaliser.

Introduction

Dans le cadre de notre dernière année à l'Ecole Nationale Supérieure de l'Electronique et de ses Applications (ENSEA), nous avons réalisé un robot roulant capable de se déplacer de manière autonome et de détecter des canettes de couleurs. Celui-ci est ensuite capable de déplacer ces canettes jusque dans une zone appropriée.

Nous avons réalisé ce projet au sein d'une équipe de 5 étudiants de spécialisation Mécatronique et Systèmes Complexes (MSC) : Ali DANDACHE, Julien MELIN, Leo PERTUET, Quentin PHILIPPE et Ludovic SCHINI. A l'initiative de ce projet, Quentin et Ali ont pris la composition de chef de projet afin de manager l'équipe et vérifier le bon avancement du projet.

L'intérêt de ce projet vient tout d'abord d'une volonté de compétiteur avec les étudiants d'une autre spécialité de l'ENSEA. En effet nous avons voulu apporter une vision revisitée du robobeer développé par les étudiants d'ESE et tenter de battre leurs robots lors des présentations des projets.

Ce projet représente néanmoins un défi pour nous puisque nous ne disposions que de 44 heures de projet contre 80 pour les étudiants de l'autre section. Nous avons également fait le choix de partir d'une page vierge et de dimensionner nos actionneurs, définir notre architecture alors que les autres étudiants disposaient déjà d'une base roulante ainsi que de capteurs définis.

D'un point de vue pédagogique, ce projet est très intéressant puisqu'il lie de nombreuses matières et nous permet de mettre en application les bases théoriques vues en cours. Nous avons ainsi pu :

- Établir un cahier des charges
- Dimensionner nos actionneurs
- Réaliser une conception Hardware du robobeer
- Lui implémenter un soft réaliser sur mesure
- Designer et fabriquer un châssis pour assembler toutes les briques du robot

La répartition des tâches a été faite selon deux critères : la compétence de la personne sur la tâche et l'envie de travailler sur le sujet. Nous avons privilégié de mettre quelqu'un sur quelque chose qu'il voulait comprendre afin qu'il apprenne quitte à "perdre du temps"... Nous sommes là pour apprendre !

Table des matières

I - DÉFINITION DU BESOIN

Le projet que nous avons décidé d'entreprendre consistait à concevoir, fabriquer et assembler un robot complet capable d'identifier et ramener une canette dans une zone prédéterminée en fonction de sa couleur. L'identification devait être réalisée par caméra et deep learning pour permettre d'identifier dynamiquement les canettes (éclairage variable, ton de couleur…).

Après deux séances consacrées au brainstorming et à l'établissement d'un cahier des charges, nous avons planifié le Gantt et nous nous sommes réparti les tâches :

- Ali à la reconnaissance d'image et codage Raspberry Pi,
- Julien à la partie alimentation et commande moteur,
- Quentin au développement des PCB, à la conception mécanique et finalement intégration et tests du software
- Ludovic à la partie capteur et communication,
- Léo à la partie capteur et suivi du projet sur redmine

Le cahier des charges était défini ainsi :

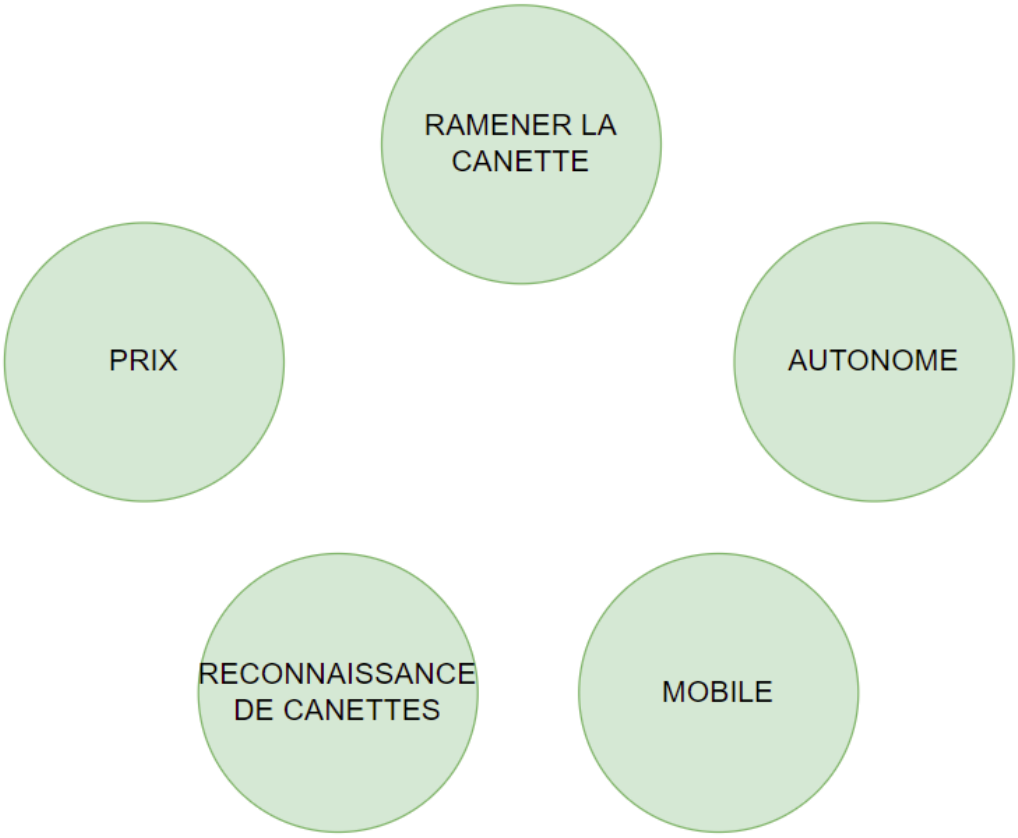


Fig1. Représentation des éléments clés du cahier des charges

Rapidement, nous avons réalisé un schéma fonctionnel du robot afin de structurer le développement.

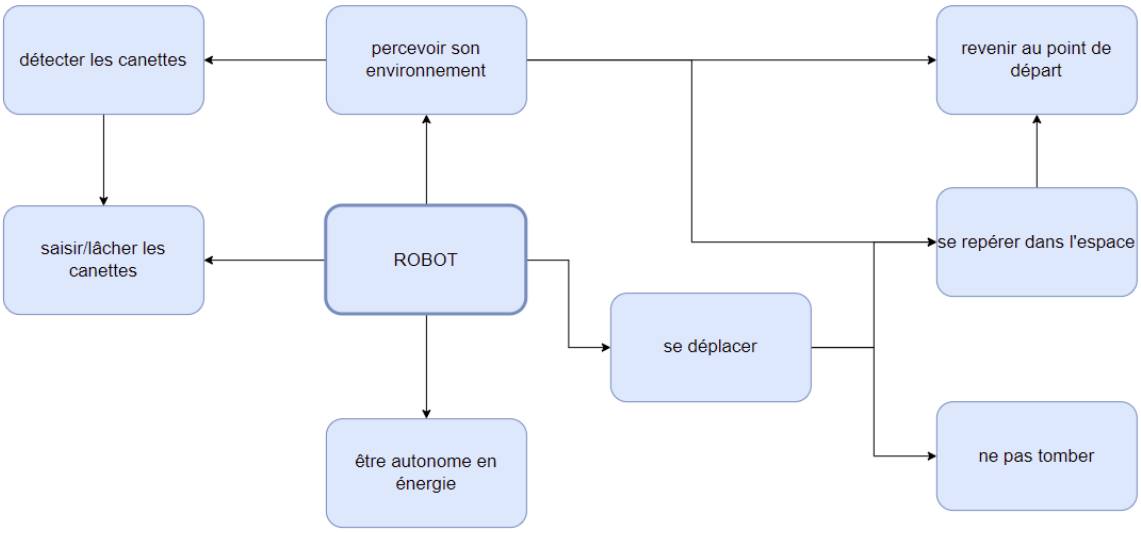


Fig2. Schéma fonctionnel

Nous avons ensuite déterminé où placer nos composants et capteurs ce qui nous a permis de commencer le développement de la base roulante.

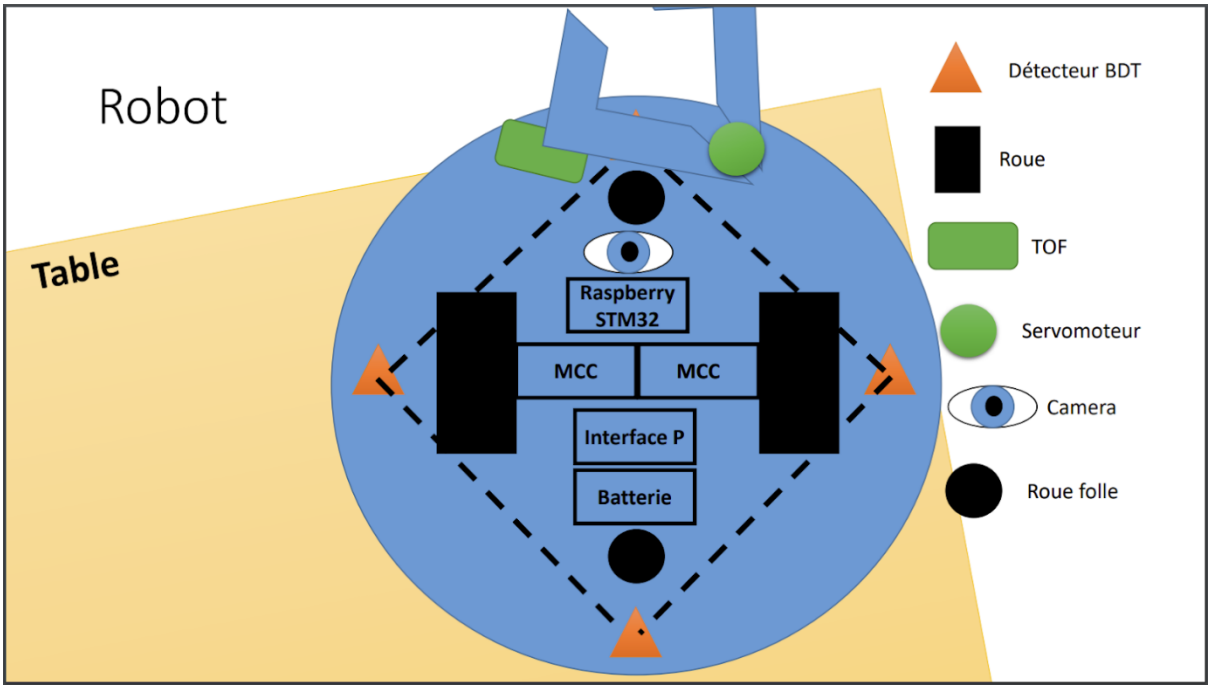


Fig3. Représentation générale du robot

Afin de simplifier la conception du robot et la rotation de celui-ci, nous avons décidé de concevoir une base roulante circulaire. En effet, à l'aide de 2 motoréducteurs placés de chaque côté et de roues folles, nous essayons de limiter au maximum les glissements mécaniques entre les roues et la surface ce qui permet de garder une bonne précision d'odométrie.

En ce qui concerne l'approche envisagée pour "capturer" les canettes, nous avons décidé d'utiliser une pince à deux doigts mobiles commandée par un servomoteur via la carte STM32.

L'affectation des tâches et le suivi des deadlines a été fait sur redmine. L'organisation et les dépendances de celles-ci ont été révisées au cours du projet afin de rendre le développement plus transparent.

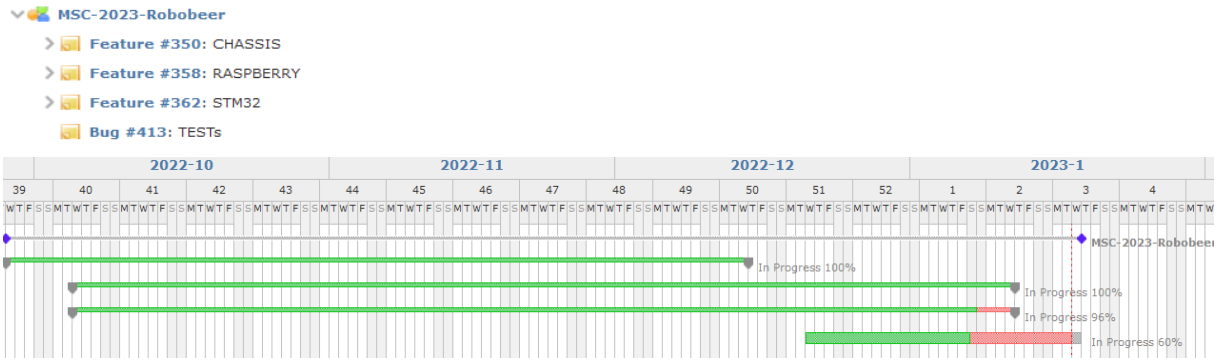


Fig4. Représentation partielle du Gantt

II- Hardware

Encodeur

L'encodeur utilisé est celui vendu en kit avec notre moteur. Il est alimenté en 3.3V ou 5V et est protégé contre les inversions de polarité. Il est composé d'un aimant permanent à 8 pôles et 2 capteurs à effet Hall nous permettant d'acquérir deux signaux carrés déphasés de 90° ce qui fait au total 16 impulsions par rotation côté moteur.

Ce moteur dispose d'un réducteur de 75:1 en sortie ce qui nous fait un total de 1200 impulsions par tour en sortie d'axe. Ainsi, nous avons 190 impulsions par radian ce qui permet une grande précision (théorique) d'odométrie.

Processeur

Différents processeurs étaient proposés pour notre projet: le STM32F103, STM32L412KBT6 et le STM32L031K6T6 (en ordre décroissant de puissance).

Nous avons déterminé assez rapidement que nous aurions besoin de nombreux I/O pour notre robot à cause des nombreux capteurs, or les deux processeurs autre que le F103 étaient limités à 25 et nous en avions besoin de 35. Bien que le F103 dispose aussi de plus de mémoire, il s'agissait du nombre d'I/O qui était surtout limitant ; nous avons donc choisi ce dernier.

Capteur Time of Flight (TOF)

Il peut être utile de mesurer directement la distance entre un objet et le robot. Le capteur utilisé est le VL53L0X produit par le constructeur ST. Pour des raisons pratiques, on utilise ce capteur déjà monté sur un PCB. Ce capteur nécessite 4 connexions, 2 pour l'alimentation et 2 pour la communication en le connectant sur le bus I2C.

Capteur Couleur

En complément de ce que peut nous fournir la Raspberry Pi comme information, on a décidé de rajouter un capteur couleur afin de vérifier la couleur de la canette une fois attrapé par la pince. Si la couleur détectée correspond à celle indiquée par la RPI, on emmène la canette dans la zone dédiée. Sinon, on se réfère à la couleur perçue par ce capteur couleur pour le choix de la zone.

Le capteur est le VEML3328. Il communiquera avec le STM32 via une liaison I2C. On a reçu que le capteur, il faut lui fabriquer la PCB selon le schéma suivant :

APPLICATION INFORMATION

Pin Connection With the Host

The VEML3328 is a cost effective solution color and IR sensor with an I²C interface. All possible settings and result data can be accessed via the standard I²C interface.

A typical application circuit is shown in Fig. 6 below. The additional 0.1 µF capacitor near the V_{DD} pin in the circuit is used for power supply noise rejection. Pull-up resistors for the I²C bus design are recommended to be 2.2 kΩ.

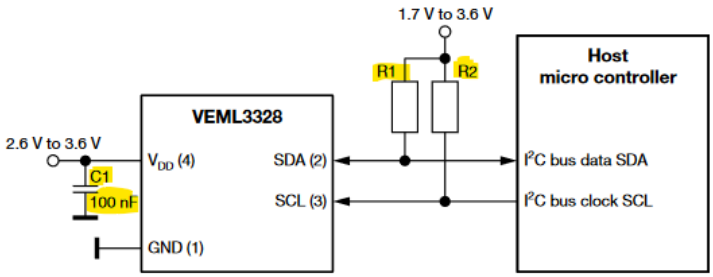


Fig. 6 - Hardware Pin Connection Diagram (Slave Address 0x10)

Fig5. Extrait de la datasheet du capteur VEML3328

On télécharge la librairie de ce capteur et on l'importe dans KiCAD.

Capteurs bord de table

Afin de détecter si on est au bord de table ou pas, on a cherché une solution peu chère. On a choisi d'utiliser des photorésistances couplées à des LEDs. L'idée est d'avoir la lumière de la LED réfléchie par la table sur la photorésistance.
Nous avons intégré sur la carte la possibilité de raccorder ce genre de système. Nous avons souhaité nous focaliser, dans un premier temps, sur autre aspect de ce projet du fait que cette solution est très peu fiable.

Capteur Caméra

Afin de détecter les canettes et commander le robot en position, nous avons décidé d'utiliser une Raspberry Pi munie d'une caméra. Le principe est "simple". On détecte les canettes à l'aide de la caméra, et on transmet les données nécessaires au STM32 via une liaison UART.

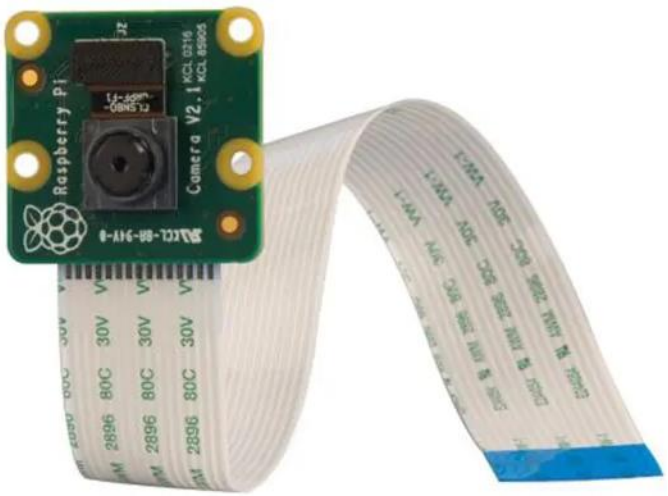


Fig6. Module Camera pour Raspberry Pi

Nous avions sous la main une Raspberry Pi 4 modèle 8Go. Nous avons utilisé cette Pi pour le traitement d'image afin de maximiser la performance.

Fig7. Raspberry Pi 4

Alimentation

Pour le dimensionnement de la batterie qui servira d'alimentation pour l'ensemble du robot nous avons tenu compte principalement de la consommation des moteurs et de la caméra, les autres éléments étant relativement négligeables.
Nous avons finalement opté pour une batterie NiMh de tension 7,2V et d'une capacité de 3.2Ah. Ce choix est justifié par le fait que cette tension nous permet d'alimenter directement les moteurs évitant ainsi l'ajout d'une partie d'adaptation. De plus, la nature de la batterie NiMh (nickel-hydrure métallique) est plus sûr qu'une batterie lithium. Enfin, sa capacité suffira largement à couvrir l'utilisation souhaitée du robot.

L'objectif de cette partie est de mettre en place une solution pour obtenir le 5V nécessaire à la Raspberry et aux divers éléments qui y sont rattachés à partir des 7,2V fourni par les batteries. On a opté pour l'utilisation d'un convertisseur buck pour sa simplicité.

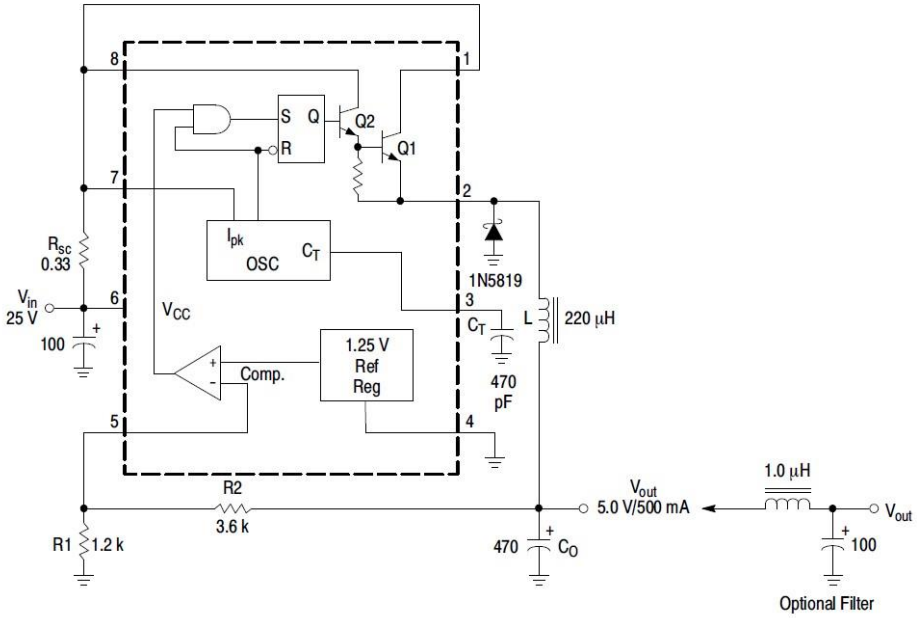


Fig7. Schéma du montage BUCK issue de la datasheet

On a donc utilisé ce montage pour réaliser un abaisseur de tension. Au vu de la possible forte demande en courant de la caméra on a décidé d'utiliser un buck pour alimenter uniquement l'ensemble raspberry + caméra et un autre pour la STM32 et les capteurs. Il ne reste alors plus qu'à effectuer les calculs permettant de dimensionner les divers composants.

Calculation	Step–Up	Step–Down	Voltage–Inverting
t_{on}/t_{off}	$\frac{V_{out} + V_F - V_{in(min)}}{V_{in(min)} - V_{sat}}$	$\frac{V_{out} + V_F}{V_{in(min)} - V_{sat} - V_{out}}$	$\frac{\lvert V_{out}\rvert + V_F}{V_{in} - V_{sat}}$
$(t_{on} + t_{off})$	$\frac{1}{f}$	$\frac{1}{f}$	$\frac{1}{f}$
t_{off}	$\frac{t_{on} + t_{off}}{\frac{t_{on}}{t_{off}} + 1}$	$\frac{t_{on} + t_{off}}{\frac{t_{on}}{t_{off}} + 1}$	$\frac{t_{on} + t_{off}}{\frac{t_{on}}{t_{off}} + 1}$
t_{on}	$(t_{on} + t_{off}) - t_{off}$	$(t_{on} + t_{off}) - t_{off}$	$(t_{on} + t_{off}) - t_{off}$
C_T	$4.0 \times 10^{-5}\ t_{on}$	$4.0 \times 10^{-5}\ t_{on}$	$4.0 \times 10^{-5}\ t_{on}$
$I_{pk(switch)}$	$2I_{out(max)}\left(\frac{t_{on}}{t_{off}} + 1\right)$	$2I_{out(max)}$	$2I_{out(max)}\left(\frac{t_{on}}{t_{off}} + 1\right)$
R_{sc}	$0.3/I_{pk(switch)}$	$0.3/I_{pk(switch)}$	$0.3/I_{pk(switch)}$
$L_{(min)}$	$\left(\frac{(V_{in(min)} - V_{sat})}{I_{pk(switch)}}\right) t_{on(max)}$	$\left(\frac{(V_{in(min)} - V_{sat} - V_{out})}{I_{pk(switch)}}\right) t_{on(max)}$	$\left(\frac{(V_{in(min)} - V_{sat})}{I_{pk(switch)}}\right) t_{on(max)}$
C_O	$9\frac{I_{out}t_{on}}{V_{ripple(pp)}}$	$\frac{I_{pk(switch)}(t_{on} + t_{off})}{8V_{ripple(pp)}}$	$9\frac{I_{out}t_{on}}{V_{ripple(pp)}}$

Fig8. Tableau de dimensionnement des composants du BUCK issue de la datasheet du composant.

On se réfère donc à la documentation du convertisseur et en suivant la colonne Step-Down on trouve:

-Pour un Vout=5V il nous faudra R1=1.2kΩ et un R2=3.6kΩ pour les 2 bucks.

-Pour la STM32: Rsc=0.5Ω ; Lmin= 10 µH ; CT=330pf ; Co=150uF

-Pour la Raspberry: Rsc=0.27Ω ; Lmin=4,7µH ; CT=330pf ; Co=300uF

Pour la diode Schottky on utilisera la STPS0520Z.

Problèmes rencontrés

La chute de tension étant assez faible, on est assez proche des limites du convertisseur.

La STM32F103 nécessitant une tension de 3.3V, il fut décidé d'ajouter un régulateur LM1177 qui fournira cette tension à partir du 5V du buck. Le régulateur permet aussi d'avoir une tension plus lisse.

Nous avons utilisé deux montages BUCKs afin de rendre indépendant l'alimentation de la Raspberry et celle des autres composants du robot. En effet, l'ensemble {caméra, Raspberry} semble consommer plus d'1A en continue. Cela aurait pour conséquence une sous-alimentation du MCU qui pourrait redémarrer en permanence.

Moteur

L'objectif de cette partie est de réaliser des tests basiques pour pouvoir piloter les moteurs. Nous avons fait le choix de piloter les moteurs via les drivers TB67H451AFNG,EL.

Table 4.1 Input and output functions

IN1	IN2	OUT1	OUT2	Mode
L	L	OFF (Hi-Z)	OFF (Hi-Z)	Stop
				Standby mode after elapsing 1 ms
H	L	H	L	Forward
L	H	L	H	Reverse
H	H	L	L	Short brake

Current path: Forward: OUT1 to OUT2, Reverse: OUT2 to OUT1

Fig9. Table I/O des drivers moteurs

Pour piloter les moteurs, on va devoir générer 4 PWM (2 pour chaque ensemble driver/moteur) qu'on fournira au driver en respectant la table ci-dessus. Les signaux d'entrées sont considérés à l'état HIGH entre 2V et 5V et LOW entre 0V et 0.8V on peut donc directement utiliser un timer de la STM. La

fréquence doit être inférieur à 400 kHz on prendra donc 20kHz pour laisser le temps au composants interne de commuter correctement.

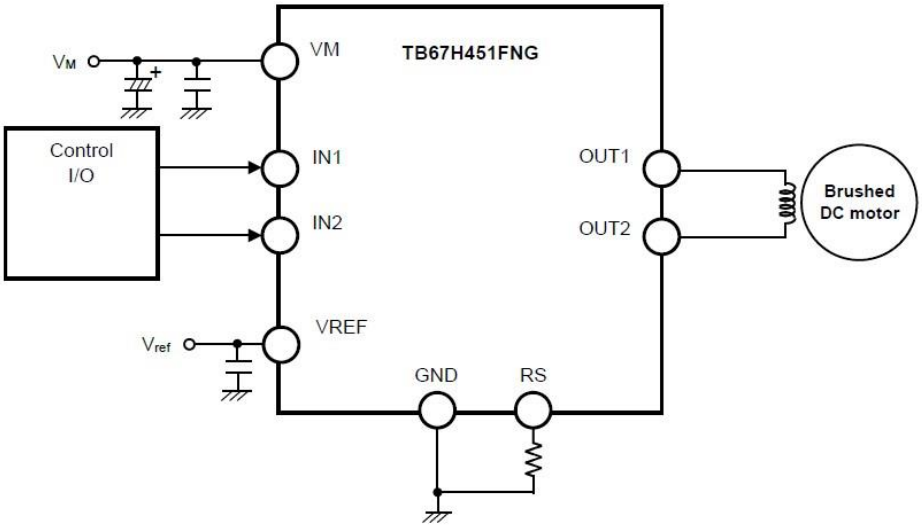


Figure 7.1 Application circuit example (Brushed DC motor)

Fig10. Montage du driver moteur

Table 7.1 Recommended capacitor values for power supply pin

Connection	Components	Typ.	Recommended range
Between VM and GND	Electrolytic capacitor	100 µF	47 to 100 µF
	Ceramic capacitor	0.1 µF	0.01 to 1 µF
(Between VREF and GND)	Ceramic capacitor	0.1 µF	0.01 to 1 µF

Fig11. Tableau de dimensionnement des composants aux environs du driver moteur

Pour utiliser les drivers il faut respecter le montage précédent. Dans notre cas on ne limite pas le courant de sortie on peut donc relier la sortie RS directement à la masse.

Problèmes rencontrés

Impossible d'avoir un signal de masse suffisamment propre malgré l'utilisation d'un petit PCB. On n'a donc pas eu la possibilité de vraiment aller plus loin de cette façon.

III - Software

Encodeur

Les données renvoyées par l'encodeur sont deux trames carrées déphasées de 90°. 1200 impulsions sont émises par tour du côté de l'axe.

Les deux encodeurs utilisent les Timers 2 et 4 du processeur. Ils sont configurés en Encoder Mode TI1 and TI2 (en rising edge) afin d'utiliser deux voies. L'ARR est configuré sur sa valeur maximale, c'est à dire 65535 afin de ne pas avoir de problème d'overflow si la vitesse est grande.

Capteurs bord de table

Une photorésistance est un composant électronique dont la résistivité varie en fonction de la quantité de lumière incidente : plus elle est éclairée, plus sa résistivité baisse. Ainsi, pour exploiter cette information, nous avons besoin d'une détection EXTI pour lire les seuils de tension.

On cherche à l'aide de la vue IOC du STM32F103 des broches EXTI disponibles pour nos capteurs bord de table. Dans le menu Pinout & Configuration, sous l'onglet GPIO, on active autant d'entrées disponibles que l'on souhaite en autorisant les interruptions.

Capteur Caméra

a. Traitement d'images

La façon la plus facile et efficace pour acquérir des images et les traiter est de se référer à la librairie OpenCV sous Python.

OpenCV est une bibliothèque libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel.

Comme nos canettes ont des couleurs et formes précises, notre première approche consistait à suivre quelques étapes de preprocessing d'image.

En premier, on utilise la libraire OpenCV pour capturer une image depuis la caméra. Ensuite, on transforme m'image de BGR (Blue Green Red) au domaine HSV (Hue, Saturation, Value).

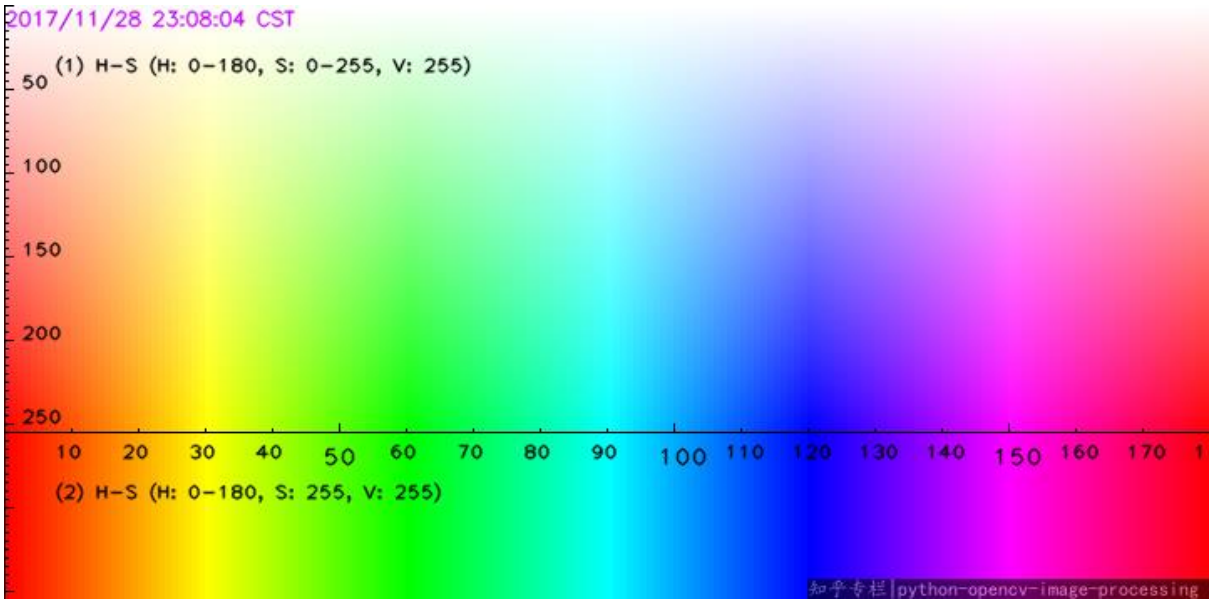


Fig12. Palette de couleurs HSV

On peut donc indiquer les valeurs de nos filtres de couleurs. Pour le Rouge et le Vert.

On remarquera que le domaine de la couleur rouge est coupé en deux, il faut donc avoir deux masques, un pour les valeurs du début du domaine et un autre pour les valeurs de fin.

Après plusieurs essaies et tuning, nous avons choisi de garder les valeurs suivantes :

```
####RED####

#lower red
lower_red = np.array([0,50,50])
upper_red = np.array([10,255,255])
#upper red
lower_red2 = np.array([170,50,50])
upper_red2 = np.array([180,255,255])
```

```
####GREEN####

#lower green
lower_green = np.array([36,30,10])
upper_green = np.array([86,255,255])
```

Fig13. Valeurs choisis pour les filtres de couleur

On applique les filtres à l'image capturé afin d'en extraire deux, une qui garde que le Vert et l'autre qui ne garde que le Rouge.

Ensuite, avec les méthodes de traitement de OpenCV, on cherche la zone de couleur la plus grande, dont le rapport hauteur/largeur entre 1.6 et 1.8. En effet une canette classique fait 115mm de hauteur et 66mm en largeur. Donc un rapport de 1.74. Cette marge a été mise ainsi afin de prendre en compte l'angle de vue, la perte de donnée (bordures de la canette) et autres aspects liés à l'optique.

Fig14. Dimensions canette classique

Enfin, il restait à déterminer le barycentre de cette zone et comparer sa position par rapport au centre de l'image. On a ainsi, la couleur de la canette et comment faire pour l'atteindre. Ces informations sont envoyées au STM32 via une liaison UART.

b. Liaison UART

Afin de communiquer avec le STM32, on a utilisé le moyen de communication le plus simple à mettre en place : l'UART. Depuis la Raspberry pi, on doit d'abord activer l'UART dans les paramètres du système. Dans un terminal, taper la commande

```
sudo raspi-config
```

On obtient la fenêtre suivante:

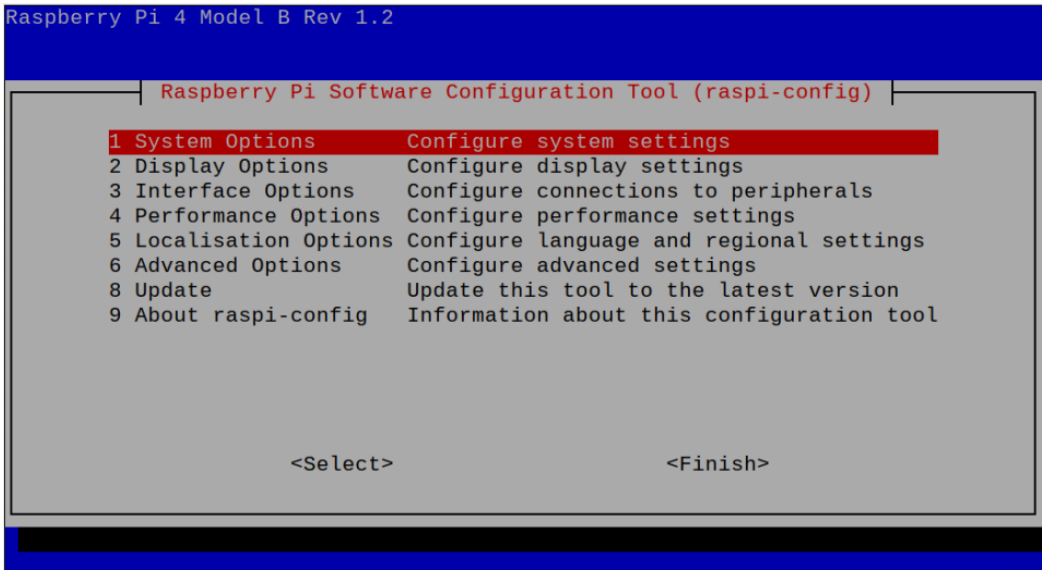
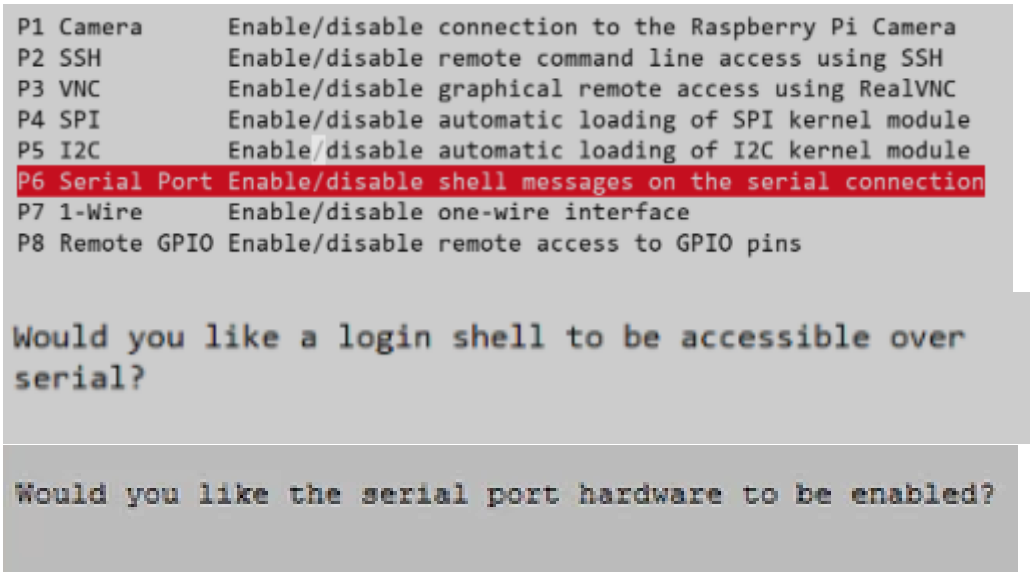


Fig15. Fenêtre de configuration de l'UART

Aller dans Interface Options, puis sélectionner non à "Voulez-vous accéder le shell via le port serial", puis sélectionner "oui" pour activer le port serial.



Le port serial est donc activer. Sur le GPIO de la Raspberry Pi, il se trouve sur les pins 8 et 10.

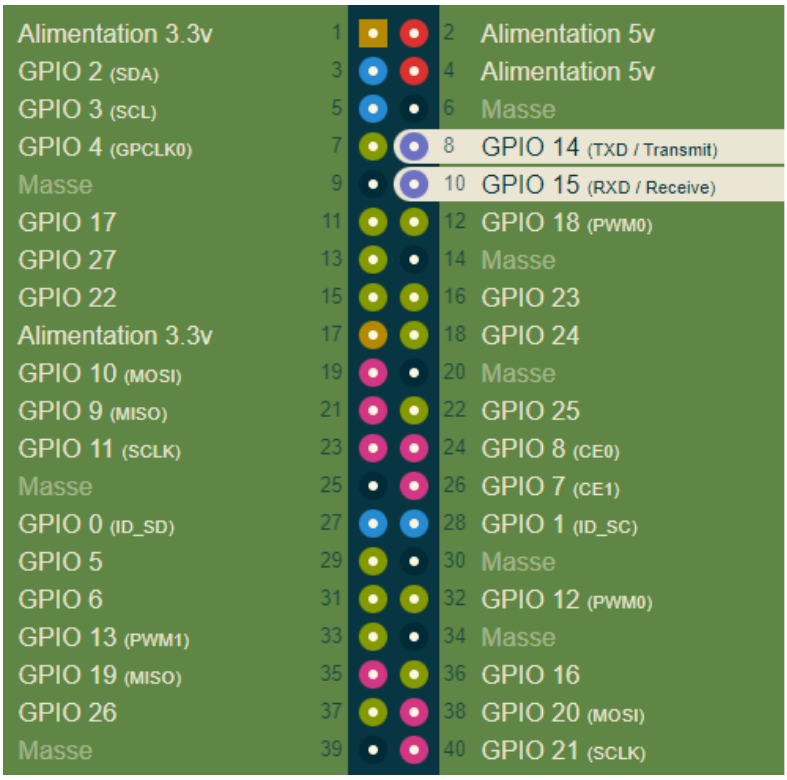


Fig16. Pinout GPIO d'une Raspberry Pi

Enfin, on peut utiliser la librairie Pyserial afin d'acceder au port serial depuis un script Python. Voici un exemple simple d'un tel script :

```
import serial

ser = serial.Serial(
   port = '/dev/ttyS0',
   baudrate = 115200,
   parity = serial.PARITY_NONE,
   stopbits = serial.STOPBITS_ONE,
   bytesize = serial.EIGHTBITS,
   timeout = 0.1,)

data = "Hello ! \r\n'
ser.write(str(data).encode()
```

La trame de communication entre la Raspberry et la carte mère est la suivante :

« FM enable action sens couleur distance RT »

Où

- RT et FM sont des caractères de début et fin de trame
- Enable est un char valant 0 ou 1
- Action est un char valant 0, 1, 2, 3
- Sens est un char valant 0 ou 1
- Couleur est un char valant 0 ou 1
- Distance est un mot de 4 char donnant la valeur de la distance estimée à parcourir

Enable	0	La camera détecte une cannette
	1	La camera ne détecte pas de cannette
Action	0	Avancer
	1	Reculer
	2	Tourner
	3	Stop
Sens	0	Droite
	1	Gauche
Couleur	0	Vert
	1	Rouge
Distance	xxxx	Distance en mm évaluée

c. Réseau de neurones

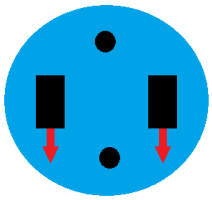
Même avec une Raspberry Pi 4 modèle 8Go, lancer un traitement d'image en temps réel pour de la détection d'objets reste une tache lourde à faire. Ceci provient du fait que le traitement sera fait par le processeur (CPU) de la Pi et non pas comme dans la plupart des déploiement dont les calculs sont parallélisés sur un GPU.

TensorFlow Lite est un ensemble d'outils conçus pour aider les développeurs à exécuter leurs modèles sur des appareils mobiles, intégrés et IoT afin d'apporter le Machine Learning sur l'appareil.

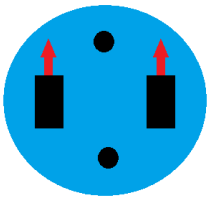
Nous avons entrainé un modèle tflite avec deux classes dedans : Canette Rouge et Canette Verte. Le modèle est basé sur l'architecture d'un modèle de base Google nommé EfficientNet-lite0. Nous avons choisi cette architecture là puisqu'elle donne le rapport fidélité/performance le plus grand possible. Notre modèle cherche dans les images les canettes et nous renvoie les coordonnées de leur Bounding Box. On peut par la suite chercher le barycentre et le comparer au centre de l'image et envoyer les commandes vers le STM32.

Moteur

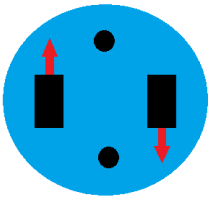
Au vus de l'architecture du robot, il nous est possible d'atteindre n'importe quelle position en n'utilisant uniquement des translations et des rotations sur soi-même.



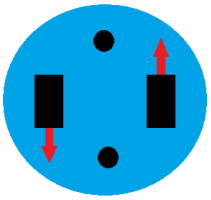
Avancer tout droit



Reculer tout droit



Rotation à droite



Rotation à gauche

En gardant la même vitesse dans les deux moteurs il nous est possible de réaliser les 4 déplacements précédents ce qui simplifie grandement le contrôle du robot.

CAPTEUR TOF

L'objectif de cette partie est de développer un système de reconnaissance des couleurs annexe à la caméra.

L'utilisation du capteur TOF se fait par le biais d'une liaison I2C. Elle se fait via une bibliothèque créée pour la carte Arduino mais à facilement été adaptée. Les fonctions suivantes ont été implémentées et permettent de communiquer avec le capteur et d'obtenir les mesures de distances :

tofReadDistance()	//Permet d'obtenir directement la distance voulue en mm sous la forme d' uint32_t.
HAL_GPIO_WritePin (ToF_Enable_GPIO_Port, ToF_Enable_Pin, SET)	//Permet d'activer le capteur ToF //Le Pin utilisé est PB1.
TofInit()	//Initialise le capteur avec les paramètres suivants: Mode 2.8V,

	Liaisons I2C Et longue distance (2m)
TofGetModel **(int *model,** **int *revision)**	//Permet de vérifier que l'initialisation du capteur s'est bien effectuée.

DATA

COM I2C

L'objectif de cette partie est d'acquérir les données des capteurs et de les communiquer au processeur afin qu'il puisse les traiter et adapter le comportement du robot.

La communication I2C est réalisée afin de communiquer entre la carte mère et les différents composants. On utilise l'I2C2 du processeur pour effectuer cette liaison avec les capteurs TOF et couleur à l'ordinateur.
Étant donné la simplicité de ce bus, il suffit de l'activer avec les paramètres de bases pour le faire fonctionner. On remarque que cela associe les pins PA8 à SDA et PA6 à SCL.

Ci-dessous une trame de base du bus I2C type utilisé.

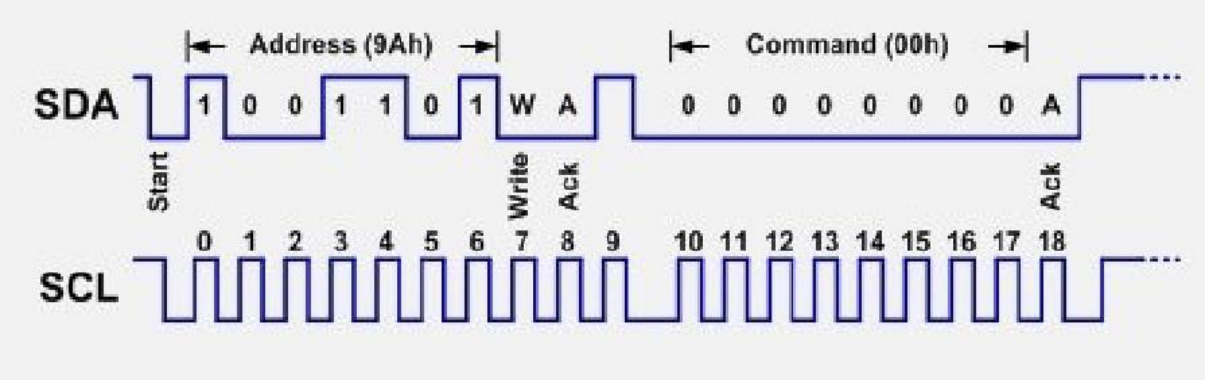


Fig17. Representation d'une trame I2C

ODOMÉTRIE

L'objectif de l'odométrie est tout d'abord de positionner le robot dans l'espace, mais aussi de prévoir ses déplacements et calculer ses erreurs afin de nous permettre de les minimiser.

Afin de contrôler la position du robot, nous découpons ses mouvements possibles en deux types différents : rotation et translation.
Les codeurs incrémentaux des moteurs s'incrémentent du même sens lorsque le robot avance, et de sens inverse s'il tourne, ce qui permet de détecter si le robot est en avance ou rotation uniquement à partir des codeurs.

Il y a 3 fonctions disponibles :

getSpeed() //Permet de récupérer la vitesse moyenne entre 100ms de mesure

getEncoderValue() // Permet d'actualiser le vecteur position du robot (x,y,angle) d'après le dernier mouvement réalisé.

updatePosition() // Permet de récupérer à la fois l'angle et la distance parcourue

On remarque que le vecteur position calculé est celui parcouru par le robot selon ses encodeurs. Cette position peut différer de la position commandée.

Enfin, on choisit de positionner le robot dans un plan cartésien avec pour origine la position de départ du robot et pour vecteur position, X, vers la droite du robot et Y, vers l'avant.

On remarque que lors de la présentation, nous disposions d'une odométrie réduite afin de pouvoir la présenter. En effet, nous utilisions la technique du "petit-poucet", c'est à dire nous enregistrons tous ses mouvements dans l'ordre, et pour revenir à la position de départ le robot effectue les mêmes déplacements mais dans un sens et un ordre opposé.

IV - Circuits Imprimés

CarteMère

Nous avions comme volonté de réaliser notre circuit imprimé nous-même et de ne pas utiliser des cartes existantes type NUCLEO ou STM32F7 pour avoir une configuration Hardware optimale pour le projet. Mais c'est également parce que c'est cool de fabriquer sa propre carte quand on est étudiant ingénieur en électronique.

L'architecture de la carte mère a déjà été prédéfinie par les choix technologiques réalisés précédemment. Notre carte dispose donc :

- D'un connecteur XT-60 pour connecter l'alimentation 7,2V
- De deux Bucks abaisseur de tension pour dissocier l'alimentation de la Raspberry et du STM32 (on souhaite avoir une tension ultra stable pour le processeur)
- D'un régulateur LDO 3.3V
- D'un processeur STM32F103
- De deux unités de commande moteur
- De connecteurs pour capteurs et actionneurs

Nous avons ajouté en plus de ces éléments un connecteur STLink pour pouvoir programmer le processeur, d'un bouton HardReset, d'un bouton UserButton ainsi que 6 Leds : 2 pour vérifier le bon fonctionnement des Bucks et pilotables via le STM32.

Les schémas électriques complets des différents blocs sont donnés en Annexe.

Concernant la taille des condensateurs et résistances nous avons travaillé au maximum avec des composants en 0603 qui sont les moins encombrants disponibles à l'école.
Néanmoins pour certaines applications on sait qu'un courant important va circuler et ces résistances 1/4W étaient sous dimensionnées.
Ainsi pour la mesure du courant de sortie des MCC nous avons utilisé des résistances 2512 qui résistent jusqu'à 1W.

Pour positionner les différents composants sur la carte nous avons défini des zones pour chaque bloc afin de minimiser la longueur des pistes.

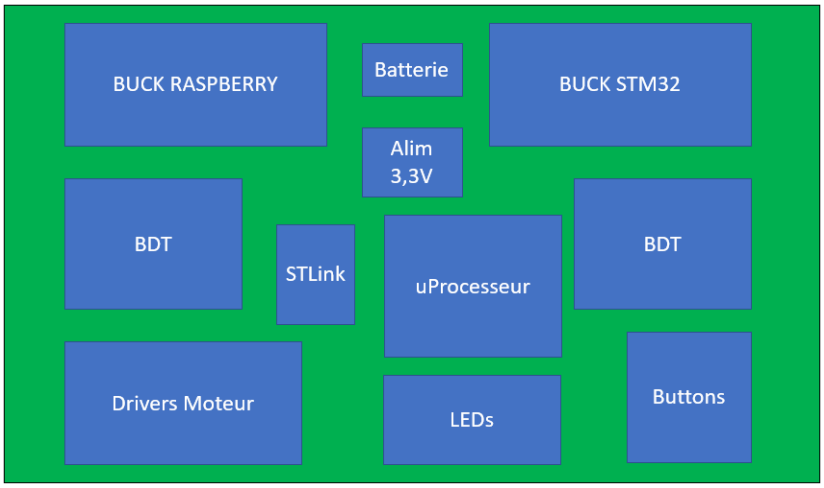
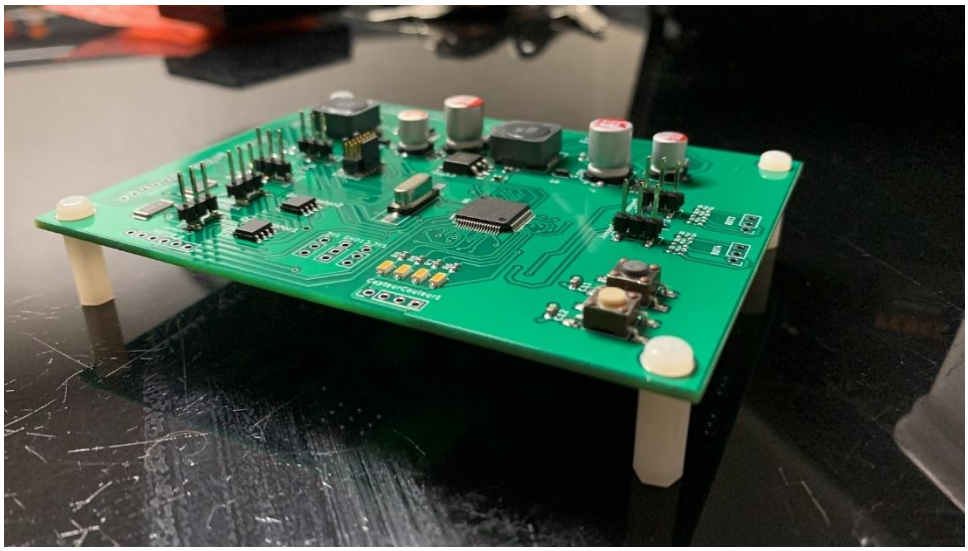


Fig18. Organisation spatiale de la carte mère

D’un point de vue CEM, nous avons réalisé un plan de masse sur le bottom et un plan 3.3V sur le top. Avec du recul nous pensons qu’il aurait été plus judicieux de mettre deux plans de masse afin de limiter la longueur des pistes.
Nous avons également positionné des capacités de découplage au plus près des composants dès lors que c’était nécessaire.

Dans un premier temps nous avons utilisé un outil d’auto-routage, le résultat peu concluant nous a forcé à réaliser toutes les connexions une par une.
Une fois le fichier Gerber créé nous avons commandé le PCB. Il s'agit d’un PCB deux couches.

Une fois le PCB reçu de Chine, nous avons soudé les composants. Les photos ci-dessous sont des photos de la carte sans les connecteurs.



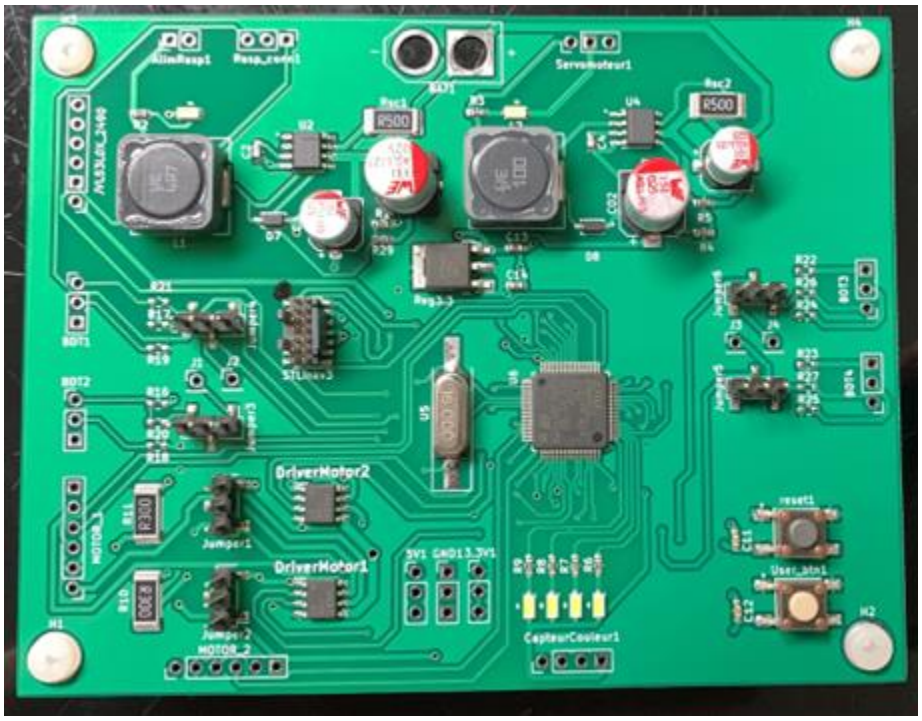


Fig19. Circuit Imprimé avec composants soudés

Capteur Couleur

Le capteur couleur a été commandé sans shield (ce qui est nécessaire pour pouvoir lire sa valeur) … Il n'était pas cher et en stock…on s'est fait avoir …

Nous avons donc réalisé un PCB sous Eagle pour pouvoir utiliser le capteur. Malheureusement suite à l'arrêt des machines de l'ENSEA, et par un manque de temps, nous n'avons pas fabriqué la seconde carte.

Selon la datasheet du capteur nous devions mettre en place le circuit ci-dessous pour pouvoir communiquer en I^2C.

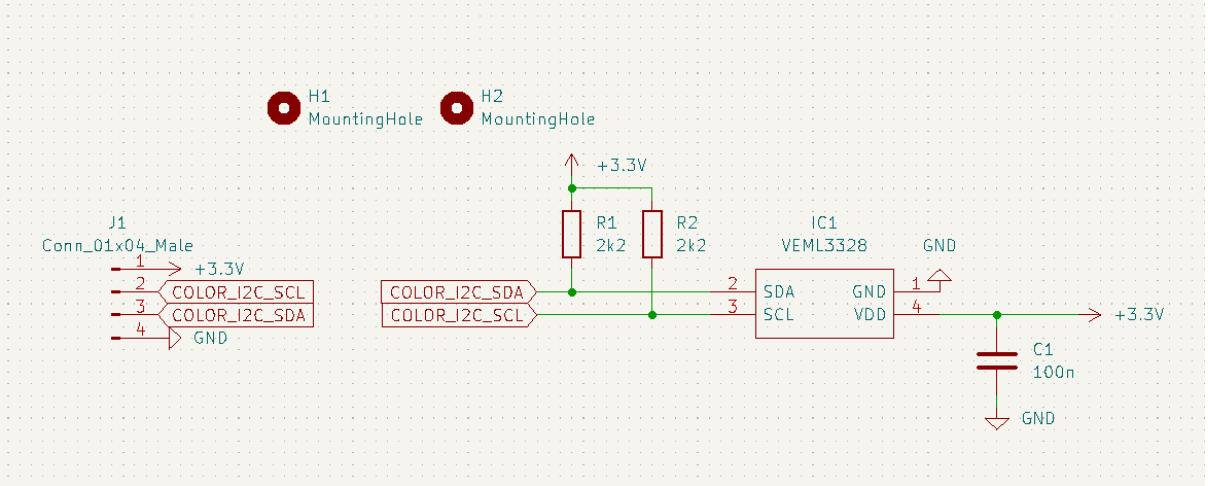


Fig20. Schéma électrique du shield capteur couleur

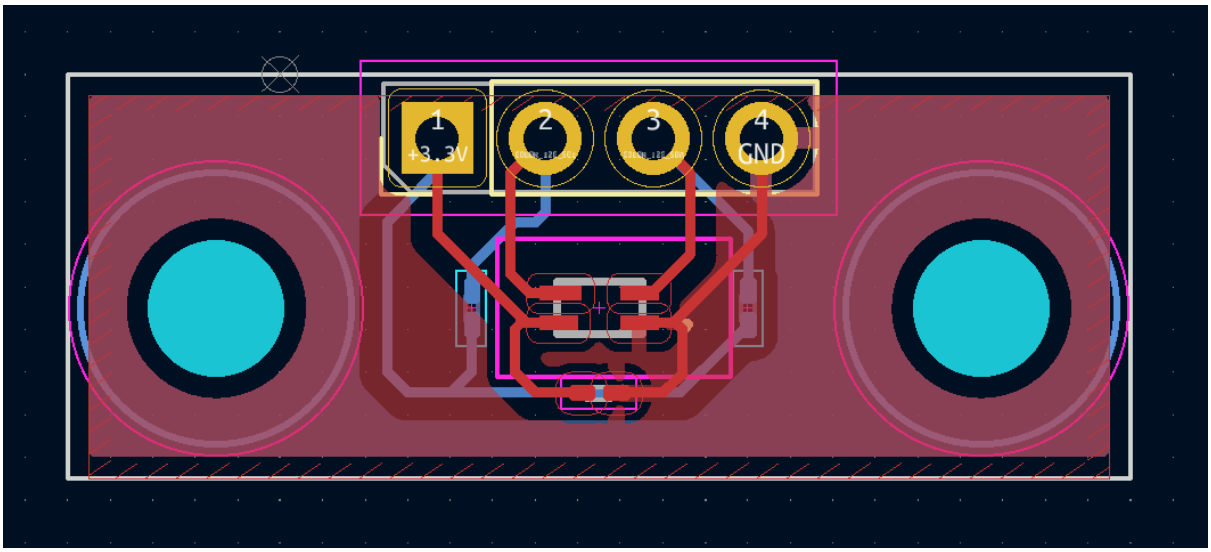


Fig21. Représentation Kicad du PCB (2.5cm de long)

Fabrication

Nous avons pu réaliser la soudure de la Carte Mère grâce à un stencil et un four ce qui a permis de souder tous les composants de la face supérieur en une seule fois.

Le process est le suivant :

- Dépôt de la pâte à souder grâce au stencil
- Dépose des composants sur les pads
- Mise de la carte dans un four et début du processus de chauffe

Dans le four, le PCB est mis à une température de 180°C puis pendant environ 40 sec la température monte de 60°C afin de faire changer d'état la pâte à souder et donc de fixer les composants de surface.
En sortie du four la carte est brûlante, il faut faire attention à ne pas se brûler et échapper la carte.
La réaction qui se produit dans le four produit des émissions nocives. Il ne faut donc pas réaliser cette opération dans un four de cuisine par exemple. (De toute façon les fours dans nos maisons ne peuvent pas monter aussi haut en température).

Fig22. Four de l' ENSEA

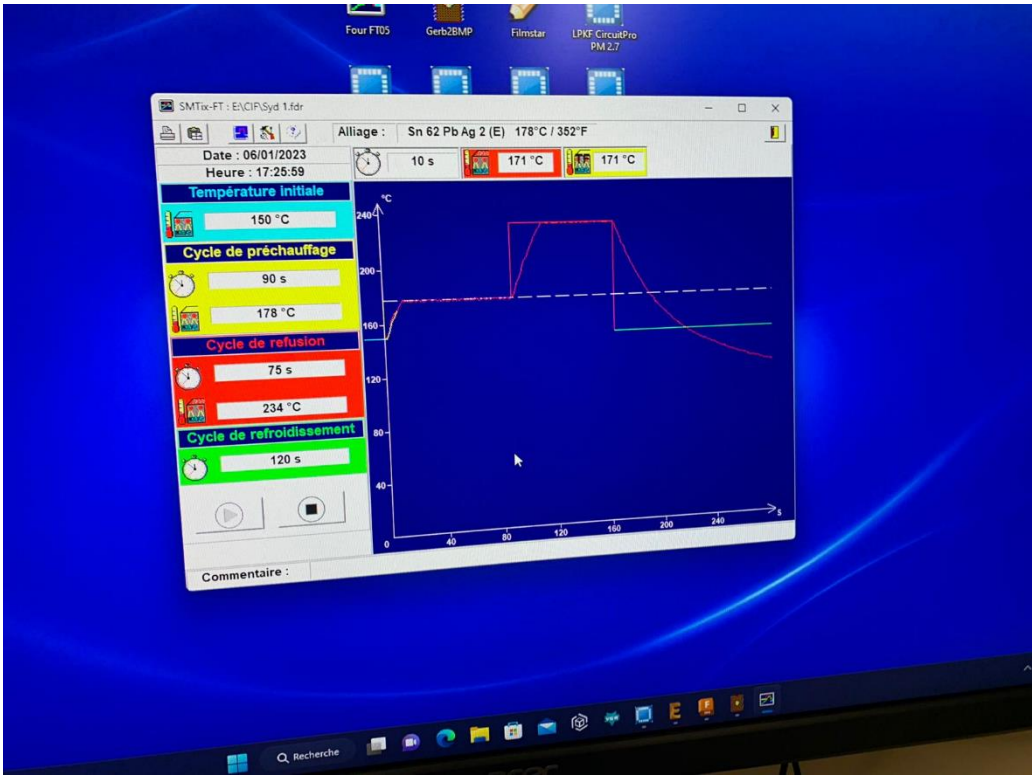
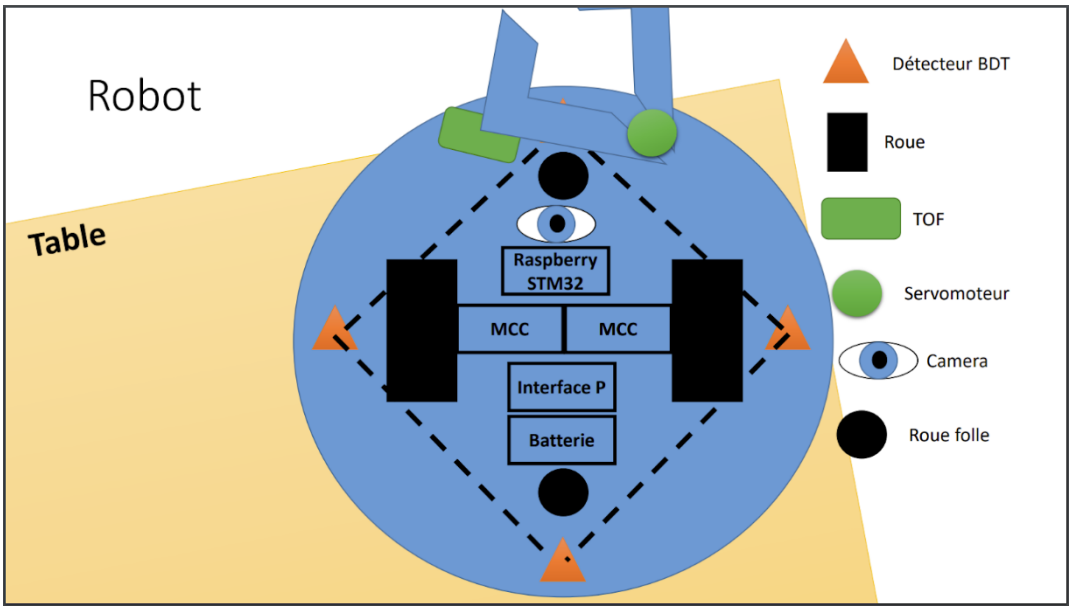


Fig23. Courbe d'évolution de la température en fonction du temps

V - Chassis

Organisation spatiale des composants (+ anticipation câble management)

Dès la première réunion de projet nous nous sommes donné une image du robot que nous allions réaliser. Il s'agit d'un robot à 4 roues : 2 motrices et 2 roues folles ayant une forme circulaire et comportant une pince.



(Rappel)Fig3. Représentation générale du robot

Physiquement il était plus simple de positionner les roues motrices au centre de la plateforme. Cette architecture permet de produire des mouvements simples faciles à réaliser : avancer tout droit, rotation. A l'aide de ces deux mouvements le robot est capable de se mouvoir dans toutes les directions

Pour équilibrer les masses nous avons fait le choix de mettre les éléments les plus lourd au centre. La batterie ainsi que les moteurs sont donc au milieu du robot. Ainsi la charge est suffisante pour réaliser l'asservissement des moteurs.

La caméra pour pouvoir mener la pince jusqu'à la canette doit préférablement être alignée à celle-ci.

Aussi nous avons décidé pour une question d'encombrement de positionner la Carte Mère ainsi que la Raspberry au-dessus de la batterie.

CAO sous SolidWorks

Nous avons décidé d'utiliser le logiciel SolidWorks. En effet ce logiciel est celui que nous avons utilisé en cours et il est disponible sur les machines de l'ENSEA ainsi que sur nos PC personnels avec la licence étudiante de l'école.

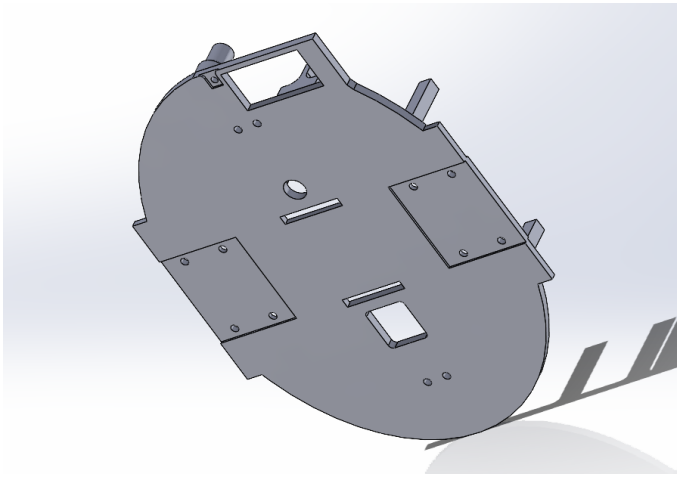


Fig24. Plateforme principale vue de dessous

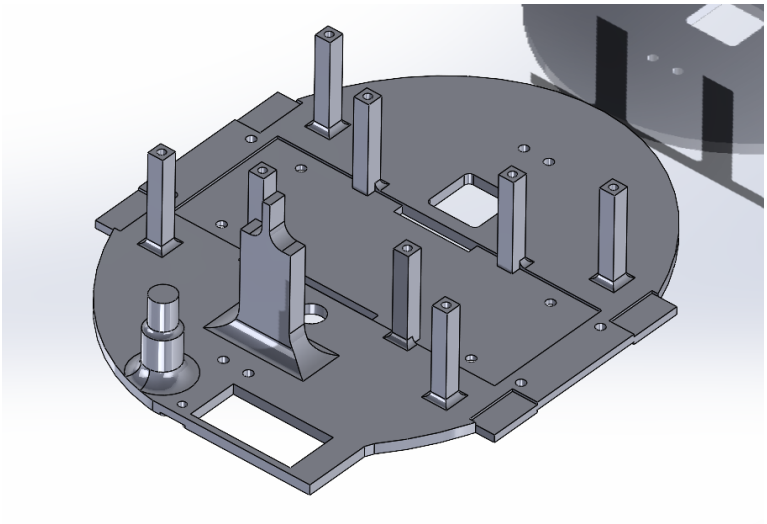


Fig25. Plateforme principale vue de dessus

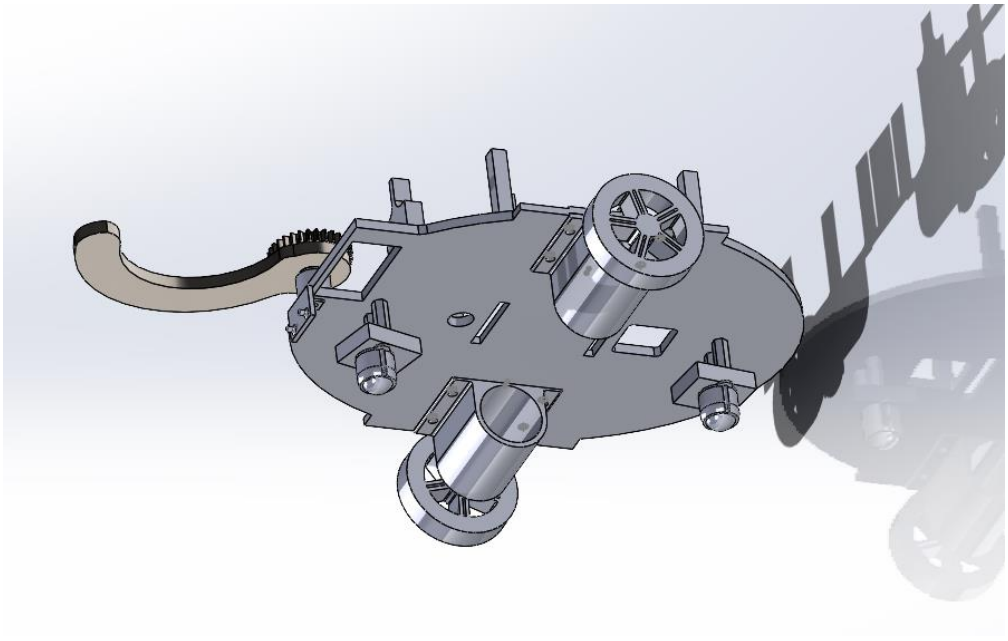


Fig26. Ensemble vue de dessous

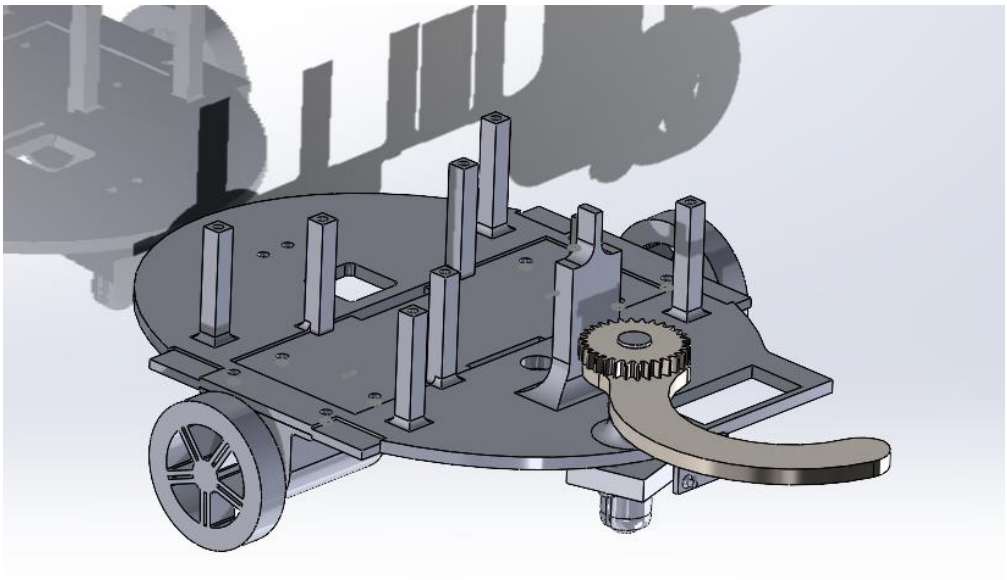


Fig27. Ensemble vue de dessus

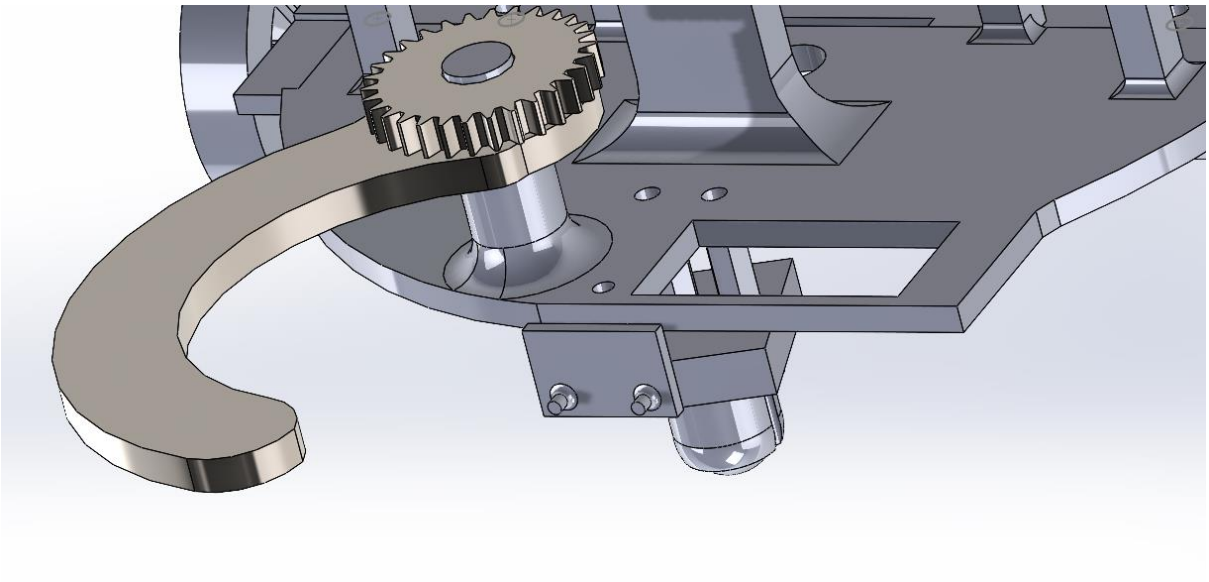


Fig28. Ensemble PINCE + TOF

Impression 3D + assemblage

Pour imprimer en 3D les différents objets que nous avons designés, nous avons utilisés le logiciel Cura. Ce slicer OpenSource permet de générer des fichiers interprétable par l'imprimante Ender3 et permet de configurer de nombreux paramètres. Nous avons choisis les suivants :

Matériau : PLA //Matière très facile à imprimer et non nocive lors de l'impression. Ce matériau se déforme dès 50°C. Pour les essais que nous réaliseront nous n'atteindrons jamais cette température

Densité de fil à l'intérieur : 20% //C'est suffisant au regard de la solidité et des efforts que vont subir les pièces

Vitesse : 50 mm/s

Ces paramètres sont déterminés empiriquement à force d'imprimer des pièces et de lire les conseils sur les forums.

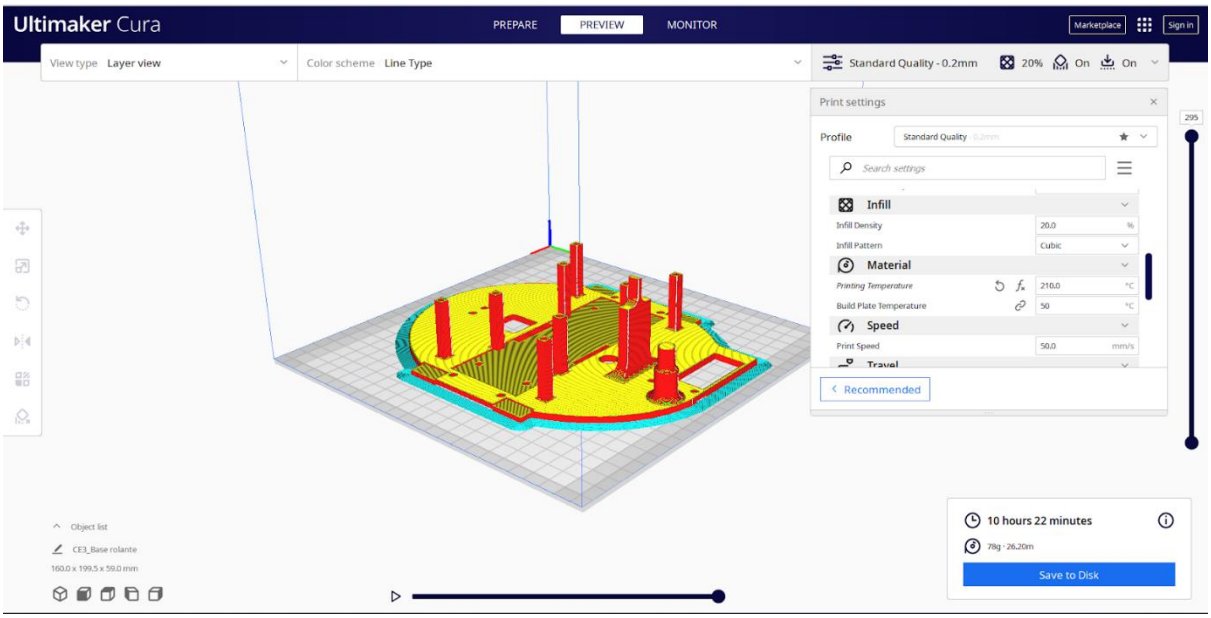


Fig29. Configuration de l'impression sous Cura

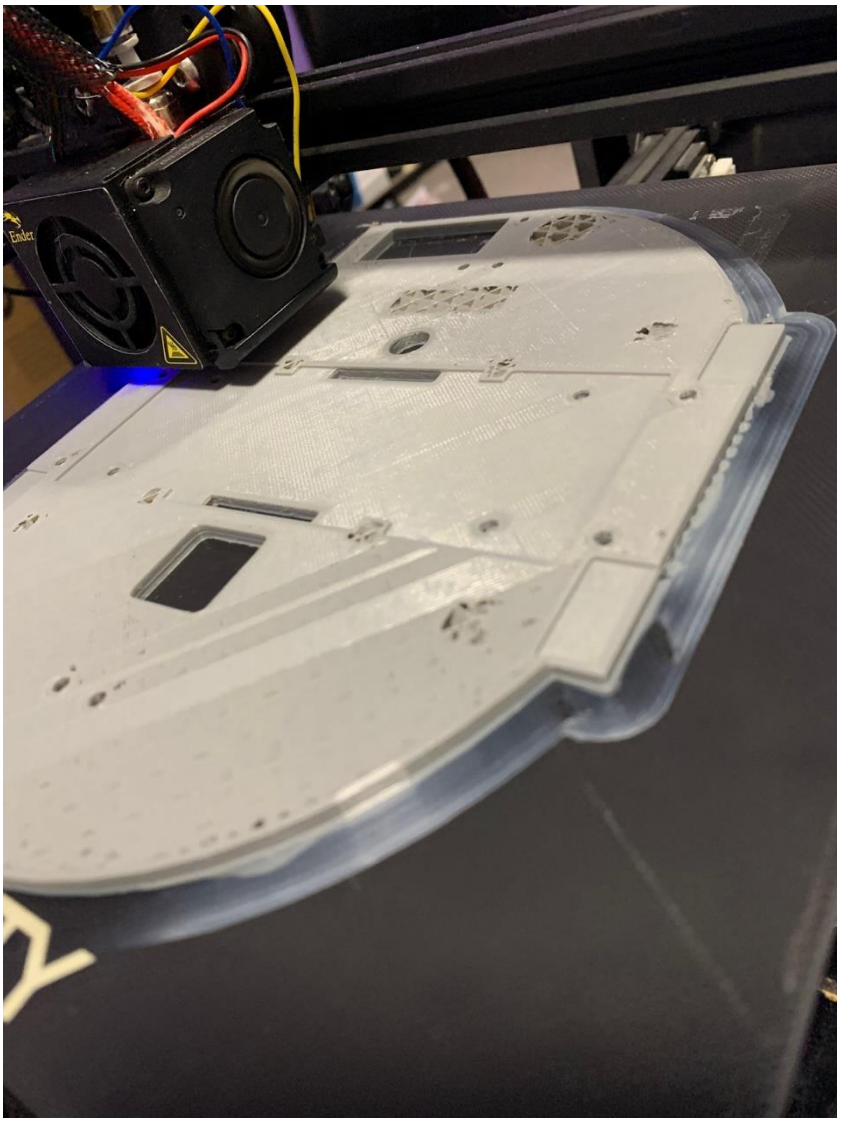


Fig30. Impression de la plateforme en 3D

Conclusion

Malgré le travail fourni durant l'année de nombreuses améliorations sont à réaliser et le projet n'est pas fini. Aujourd'hui le robot est capable via le capteur TOF de détecter une canette et de s'approcher d'elle. Il lui arrive d'attraper correctement la canette comme de la heurter.

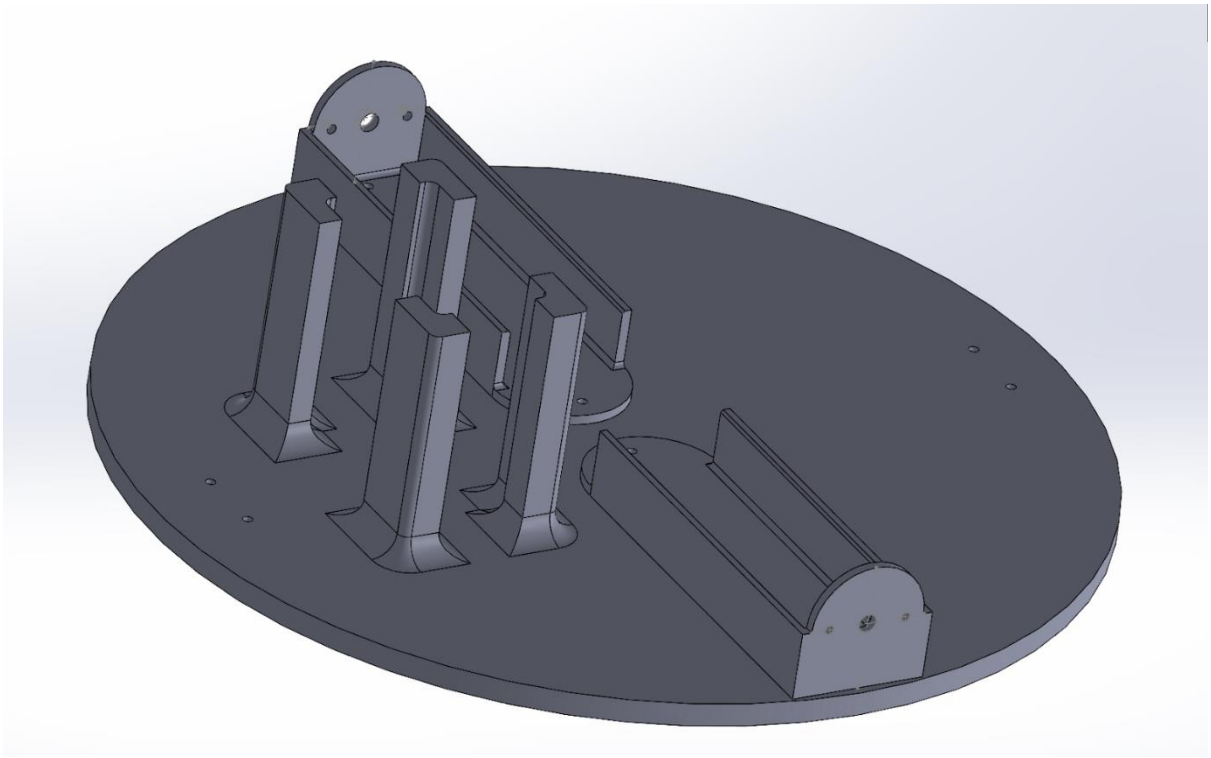
Le fonctionnement de la Raspberry a été éprouvé. Sa communication via UART (en utilisant le DMA de la carte mère) est lui aussi fonctionnel. Néanmoins nous n'avons pas réussi à alimenter la Raspberry depuis la carte mère la rendant donc non-utilisable pour le moment. En effet la carte délivre 4.95V contre 5.1V attendu. Il suffit de changer la valeur du pont diviseur de tension pour régler ce souci.

Un unique capteur Bord de Table a été monté sur le robot afin de détecter s'il s'approchait en marche avant d'une extrémité. Il fonctionne mais nécessité d'être calibré selon la luminosité ambiante dans la pièce. On pourrait améliorer le processus grâce à un capteur infrarouge par exemple.

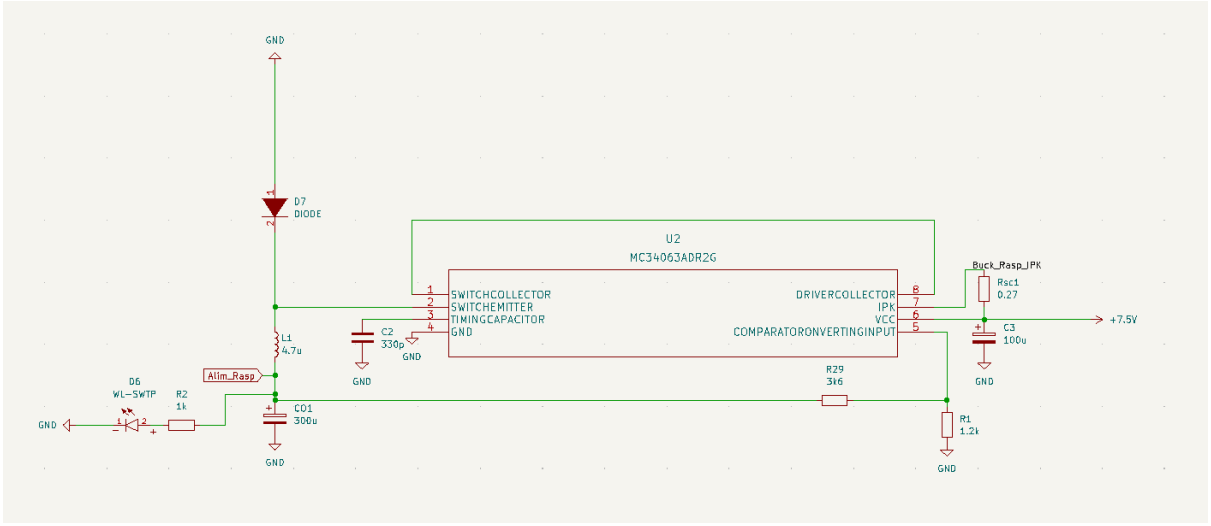
Si nous devions recommencer ce projet, l'organisation serait différente. Nous attacherions beaucoup plus d'importance à obtenir un châssis ainsi que les PCB en des délais très courts. Du point de vue software nous aurions gagné du temps si les fonctions développées en amonts étaient « mieux codées » grâce à l'utilisation de pointeurs en entrée pour les rendre indépendantes du Hardware.

Ce projet aura été riche en apprentissage. Partir d'une feuille blanche pour réaliser de A à Z toutes les étapes de conception et de tests d'un projet est réellement quelque chose de passionnant. Nous avons donc pris beaucoup de plaisir à réaliser ce robobeer et n'avons pas compter les heures. C'est pourquoi nous souhaitons continuer ce mini-projet très formateur.

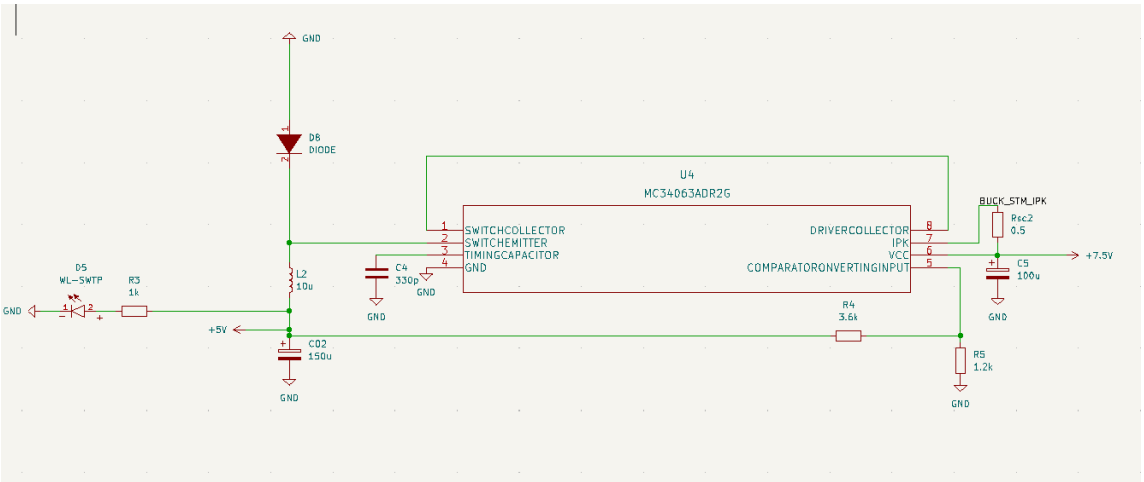
Annexe



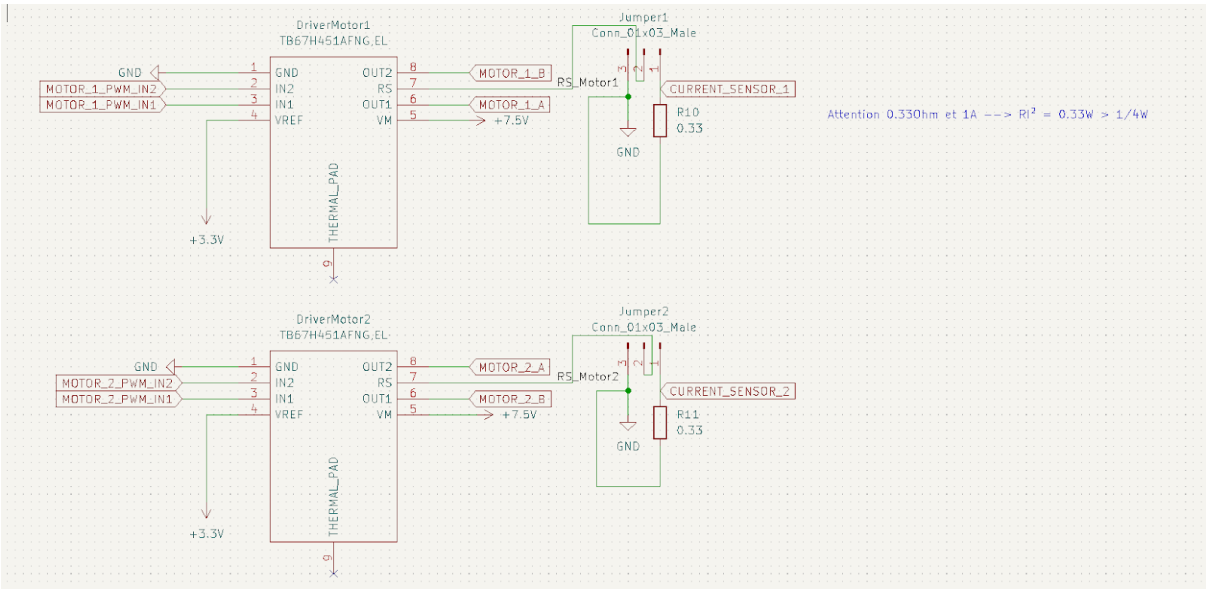
Annexe1. Premier prototype du châssis



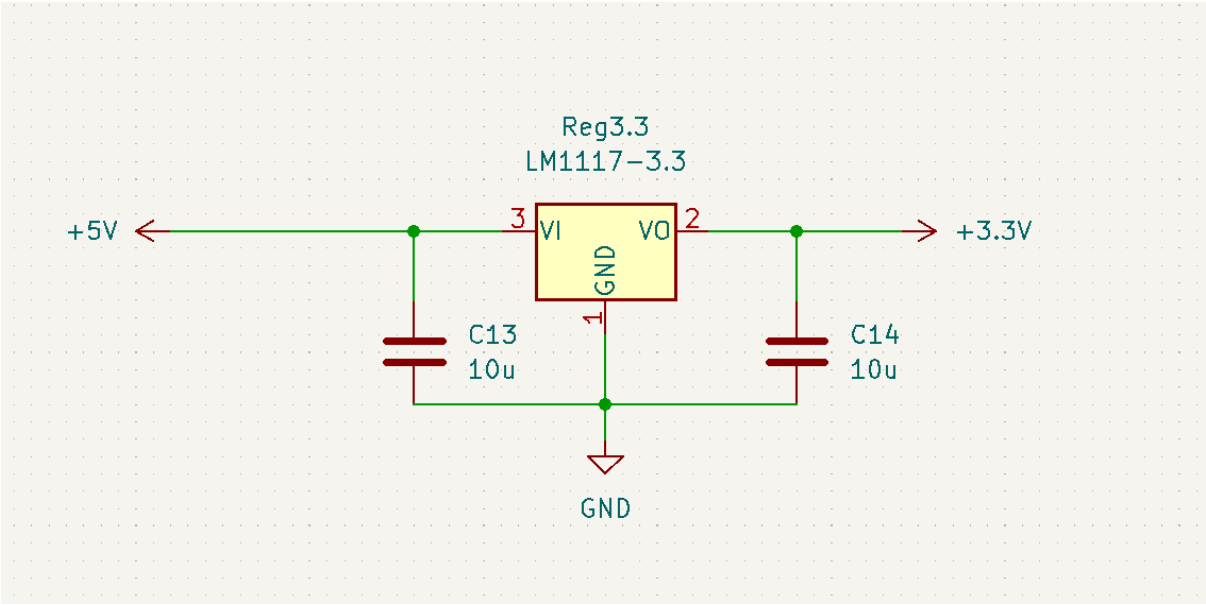
Annexe2. Schéma du montage Buck Raspberry



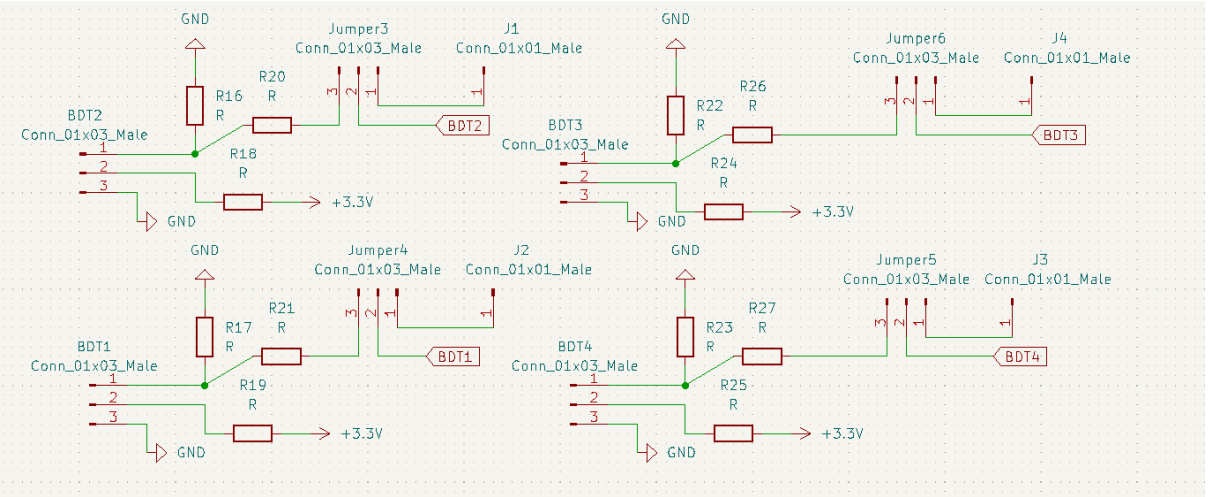
Annexe3. Schéma du montage Buck STM32



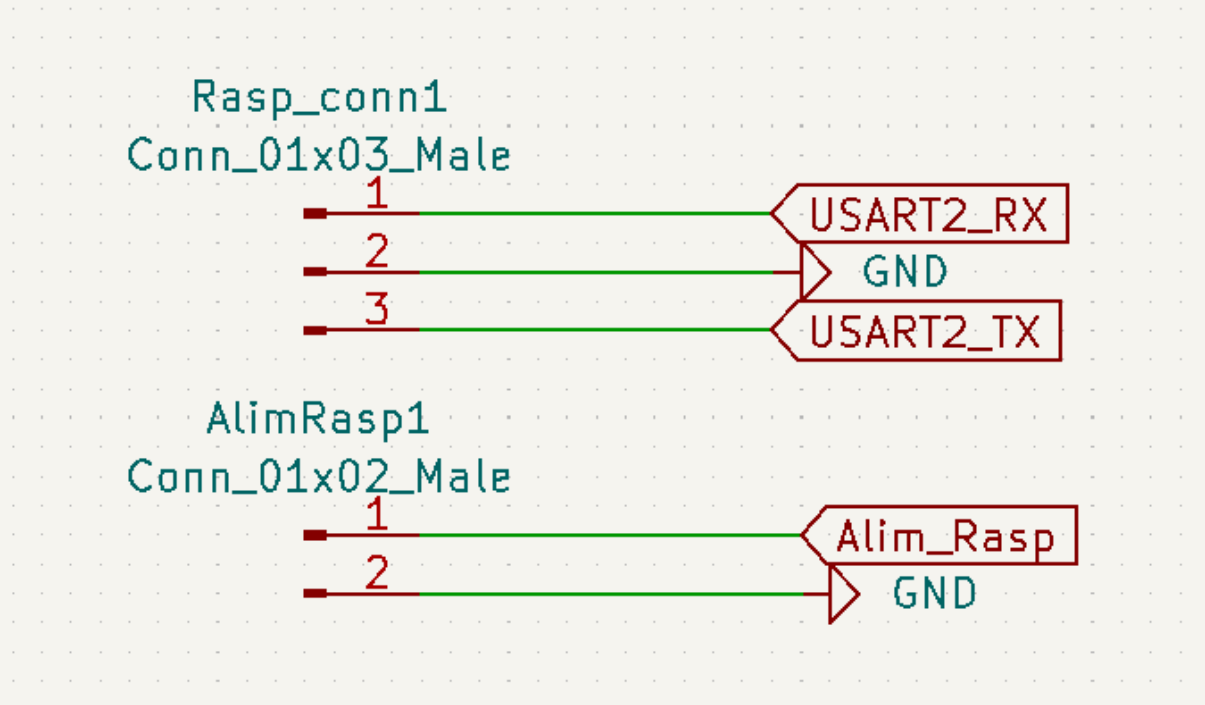
Annexe4. Montage électrique des Drivers Moteurs



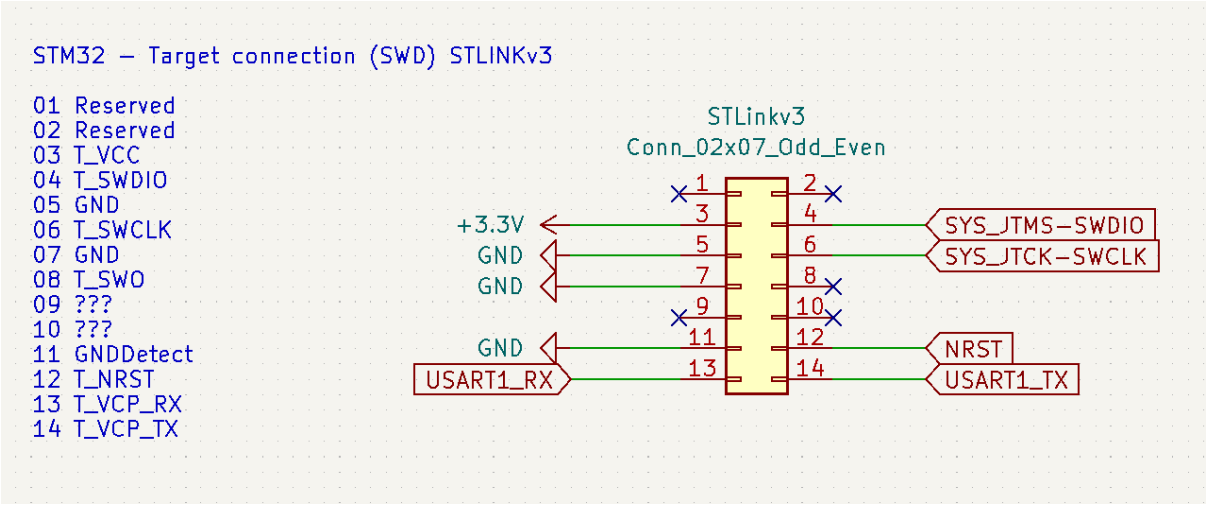
Annexe5. Montage du régulateur 3.3V



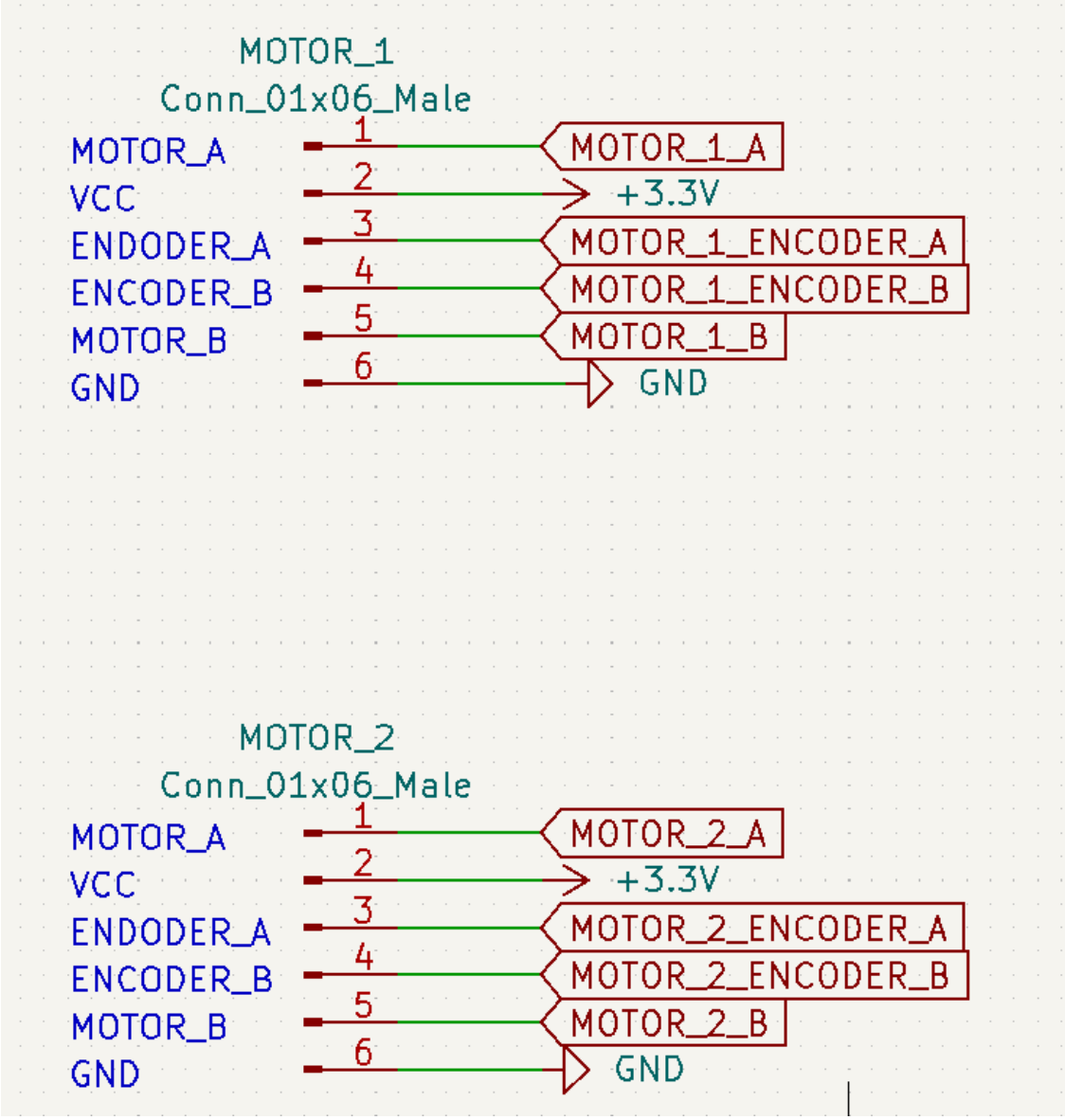
Annexe6. Montage des Capteurs Bord De Table



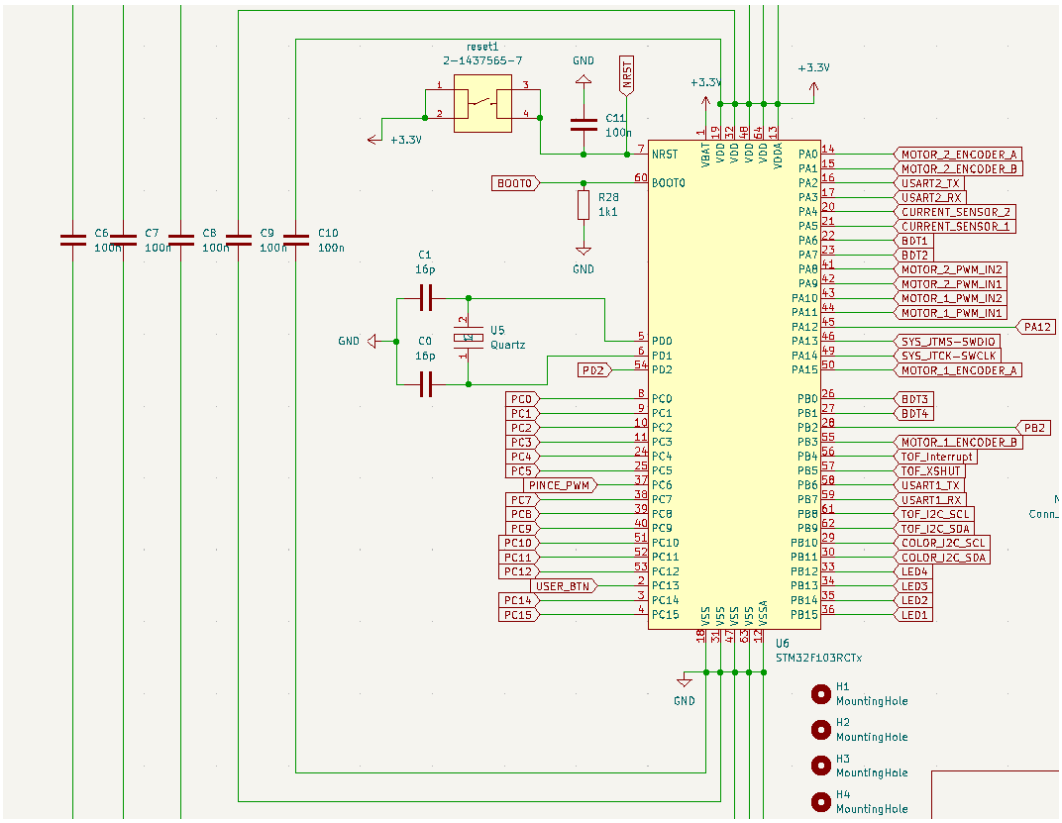
Annexe7. Connection entre la Raspberry et la STM32



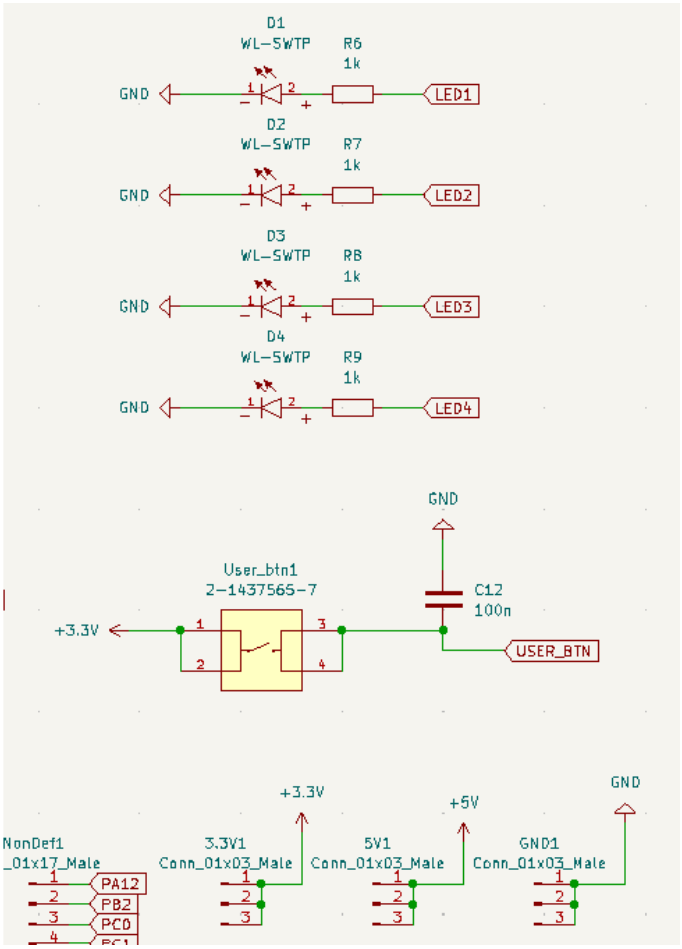
Annexe8. Connection du STLink v3



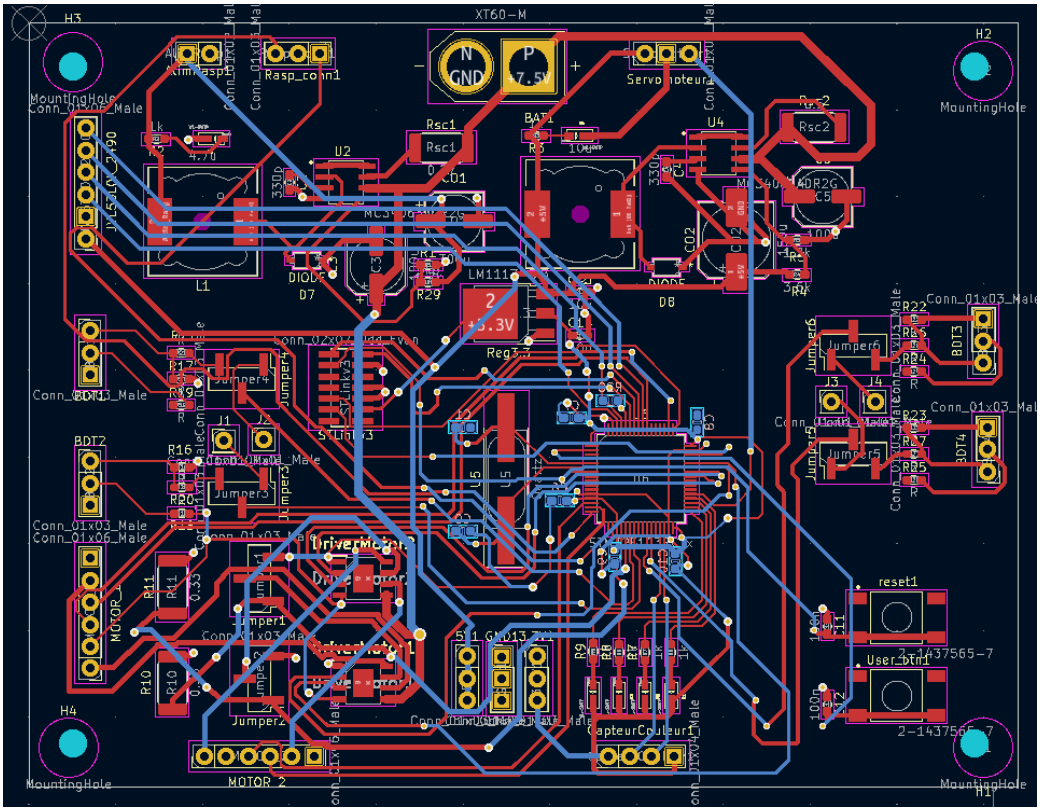
Annexe9. Connecteurs MCC



Annexe10. Montage de la STM32

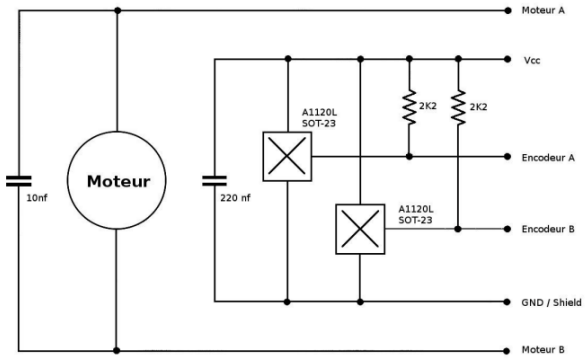


Annexe11. Autres composants (Alimentations, buttons, LEDs)



Annexe12. Représentation du PCB sous Kicad

Motoréducteur	**WT341**	**WT751**
Alimentation moteur	6 Vcc (7,5 Vcc maxi)	
Vitesse à vide	295 tours/min	133 tours/min
Consommation à vide	0,35 A	
Consommation moteur bloqué	5,5 A	
Couple moteur bloqué	4,0 kg.cm	8,8 kg.cm
Rapport de réduction	34:1	75:1
Axe	Ø4 x 10 mm avec méplat	
Dimensions	Ø25 x 77 mm (axes inclus)	
Résolution encodeur	16 impulsions/tour moteur 544 imp./tour sortie d'axe	16 impulsions/tour moteur 1200 imp./tour sortie d'axe
Alimentation encodeur Vcc	3 – 24 V	
Sortie A	Signal carré GND-Vcc	
Sortie B	Signal carré GND-Vcc	



Annexe13. Datasheet du Motoréducteur (ENCODEUR WT751)