

- Rapport de projet thématique S8 -

**AUTOMATISATION DU DÉPLACEMENT D'UNE PLATEFORME
MOBILE - ROBOT MECANUM À SUSPENSIONS**



ENSEIRB-Matmeca
Bordeaux - Talence

Encadrant : P. Melchior

Table des matières

1	Introduction	3
2	Présentation du projet	4
2.1	Scanner Laser 360° RPLiDAR A2M8	4
2.2	Caméra Intel Realsense T265	5
2.3	Robot mecanum avec suspensions	6
3	Prise en main	8
3.1	Matériel	8
3.2	Pré-requis	8
3.3	Prise en main	9
3.4	Fonctionnement et commande de l'Arduino	10
4	Objectifs	11
5	Améliorations	11
5.1	Au niveau du code	11
5.2	Mise en place d'une interface	11
5.3	Fabrication d'un support	12
6	Problèmes rencontrés	15
6.1	Câblage des sorties de la carte Arduino	15
6.2	Incendie de la batterie Lipo	15
7	Pistes d'améliorables futures	16
8	Conclusion	17
9	Annexes	18
1	Caractéristiques techniques	18
1.1	RPLiDar A2M8	18
1.2	Caméra Realsense T265	18
1.3	Code de la fonction main() du fichier rcup_finale.py	19
1.4	Code de la fonction run() du fichier rcup_finale.py	21

1 Introduction

Ce rapport s'inscrit dans le cadre du projet thématique que nous devons réaliser au S8, en filière électronique à l'ENSEIRB-Matmeca.

Notre projet consistait en l'automatisation du déplacement d'un robot à roue mecanum avec suspensions, ou plus particulièrement en l'amélioration de son déplacement. En effet, un ancien élève avait réalisé son stage de deuxième année sur celui-ci. Nous devions donc prendre la suite et y apporter les améliorations qui nous semblaient pertinentes.

Le fonctionnement du robot est le suivant : il s'agit d'un robot capable de se déplacer de façon autonome dans un environnement complexe (composé d'obstacles), et capable de les esquiver. Après avoir défini comme consigne un certain point d'arrivée, de par ses coordonnées, le robot est en mesure de se déplacer dans l'espace préalablement scanné, afin d'atteindre cet objectif. Nous allons voir dans un premier temps comment le robot effectue cette tâche, puis dans un second temps, comment nous avons fait en sorte de l'améliorer.

Ce robot devrait participer à la compétition de robotique RCup qui se tiendra à Bordeaux en 2023.

2 Présentation du projet

Le système sur lequel nous avons travaillé était donc composé du robot mecanum avec suspensions de la marque Robot-Maker, mais également d'un scanner laser 360° RPLiDar A2M8, et d'une caméra Intel Realsense T265. Chacun de ces éléments est décrit ci-dessous.

2.1 Scanner Laser 360° RPLiDAR A2M8



FIGURE 1 – Scanner Laser 360° RPLiDAR A2M8

Le Scanner Laser 360° RPILDAR A2 appartient à la nouvelle génération de lidars 2D à 360 degrés. Conçu par la marque SLAMTEC, il adopte un système de mesure de triangulation (voir figure ci-dessous) et a d'excellentes performances dans tous les types d'environnements intérieurs et extérieurs, sans exposition directe au soleil.

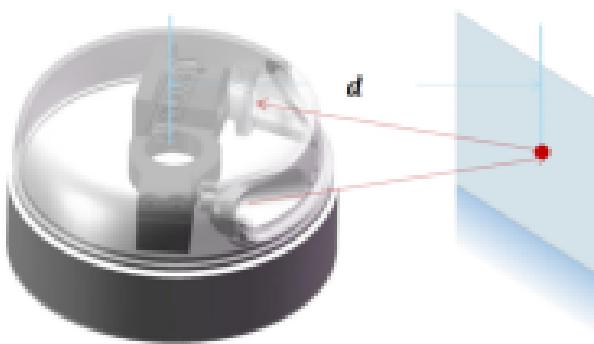


FIGURE 2 – Système de mesure du Scanner Laser 360° RPLiDAR A2M8

Il est capable de prendre jusqu'à 4100 échantillons par secondes grâce à une vitesse de rotation élevée. Sa fréquence de balayage est de 10 Hz soit 600 tr/min. Il peut effectuer un balayage 2D à 360 degrés dans une plage de 8 mètres. Chaque élément appartenant à cette zone est détecté puis assimilé à un point qui correspondra à un obstacle.

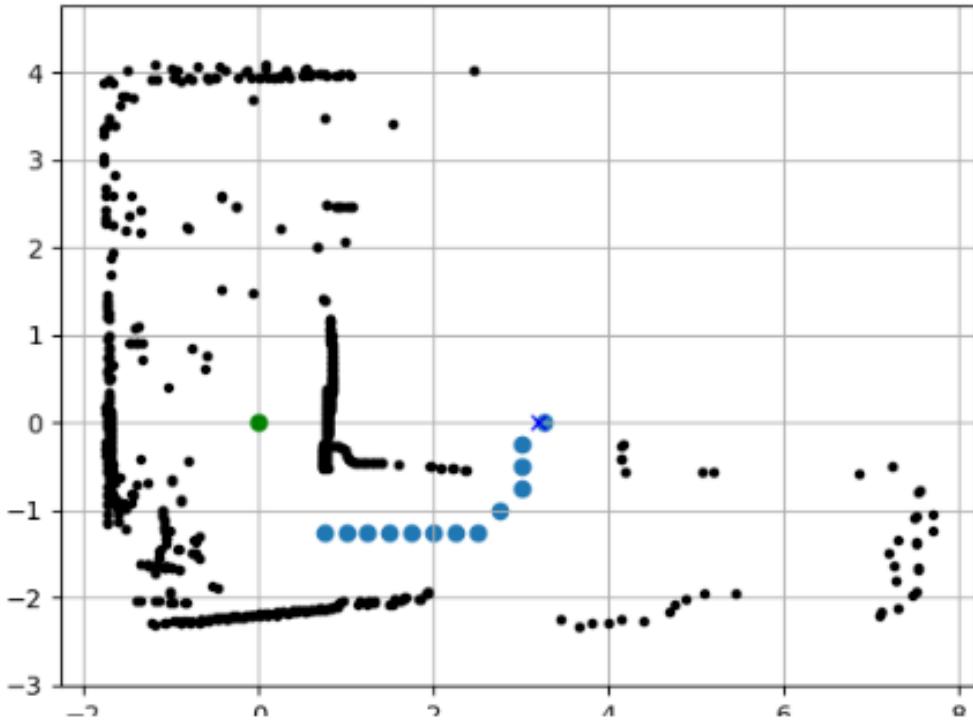


FIGURE 3 – Exemple d'une map de salle de TP

On peut y apercevoir la position d'origine (point vert), le point à atteindre (croix bleue) associée à certaines coordonnées, ainsi que le trajet que va suivre le robot pour l'atteindre (points bleus).

2.2 Caméra Intel Realsense T265



FIGURE 4 – Intel Realsense Caméra Tracking T265

Le suivi de la caméra est principalement basé sur les informations recueillies à partir de deux capteurs à objectif fisheye embarqués, chacun avec un angle de vision d'environ 163 degrés (± 5 degrés) et effectuant une capture de 30 images par seconde. Le large champ de vision de chaque capteur permet de garder les points de référence visibles pour le système pendant une période relativement longue, même si la robot se déplace rapidement dans l'espace.

2.3 Robot mecanum avec suspensions

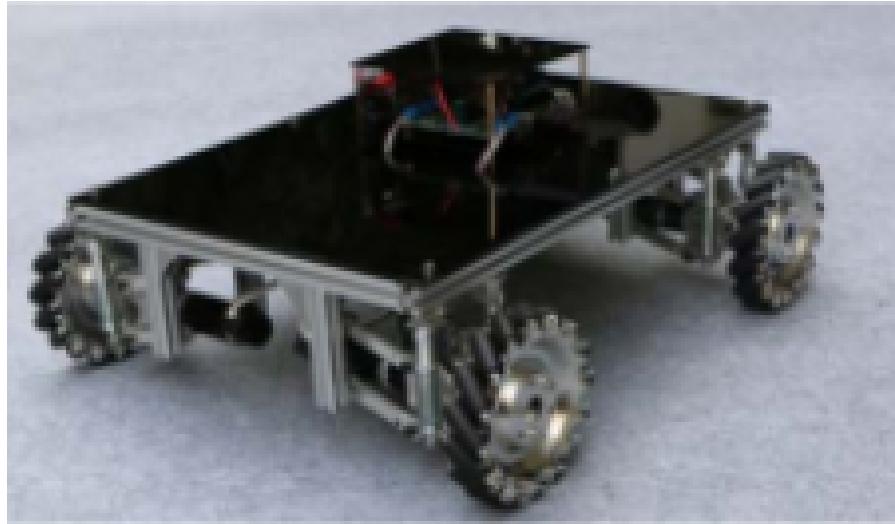


FIGURE 5 – Robot mecanum avec suspensions

Le robot est composé d'une plate-forme posée sur 4 roues mecanum, chacune contrôlée par un moteur 24V-35W. Ces 4 moteurs sont montés sur des suspensions indépendantes permettant de garder une bonne répartition de la charge sur les 4 roues, qui peuvent ainsi rester en permanence en contact avec le sol, même sur des terrains non plat. Ainsi, le robot peut se déplacer dans toutes les directions et effectuer des rotations selon l'axe vertical. Le robot possède un boîtier de contrôle avec un bouton ON/OFF et un bouton d'arrêt d'urgence. Globalement, le robot est alimenté par une batterie LiPo. Robot-Maker a tout fait normalement

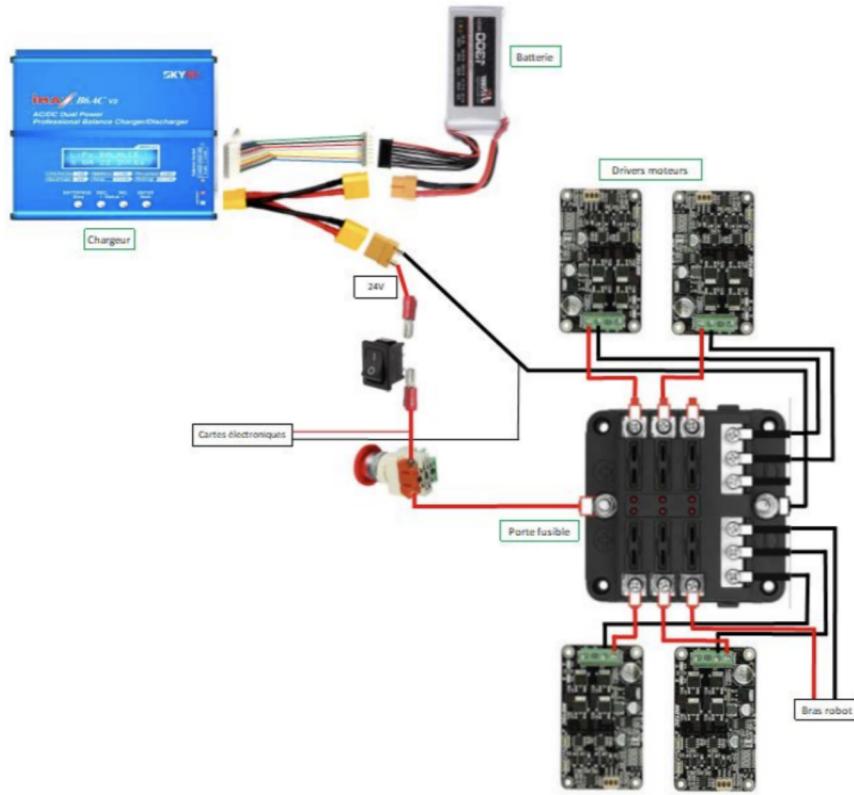


FIGURE 6 – Schéma de câblage de la partie puissance du robot

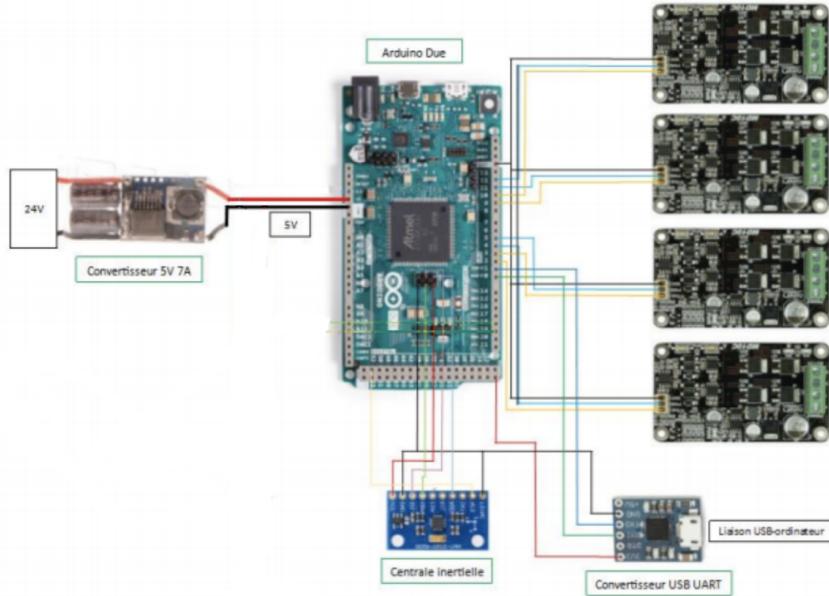


FIGURE 7 – Schéma de câblage de la partie commande du robot

3 Prise en main

3.1 Matériel

- Ordinateur avec Windows 10
- Robot avec roues mecanum
- Caméra Intel RealSense T265 et drivers.
- Visual Studio Code (ou équivalent)
- Le logiciel Arduino IDE
- Les bibliothèques nécessaires (pip, pyrealsense2, matplotlib.pyplot, serial, rplidar, time, numpy, math)

3.2 Pré-requis

Comme le langage de programmation choisi est Python, il faut installer la version 3.7.6 de Python pour des raisons de compatibilité de bibliothèque. Le lien de téléchargement : <https://www.python.org/downloads/>. Il est aussi conseillé d'installer pip (lors de l'installation de Python) afin d'installer les différentes modules/-bibliothèques mentionnées ci-dessus. Pour plus d'informations sur l'utilisation de pip : <https://python.doctor/page-pip-installer-librairies-automatiquement>.

Afin que le robot puisse interpréter les commandes qu'il reçoit et les réaliser, il faut flasher la carte Arduino Due avec le code de déplacement fournie par le constructeur du robot. Cela en ouvrant le fichier `deplacement.ino` avec l'Arduino IDE puis en téléversant le code à la carte.

Mode d'emploi pour flasher la carte Arduino :

Il faut d'abord ouvrir le fichier `deplacement.ino` avec le logiciel Arduino IDE (<https://www.arduino.cc/en/software>). Il faut ensuite se connecter sur le port de programmation de la carte.

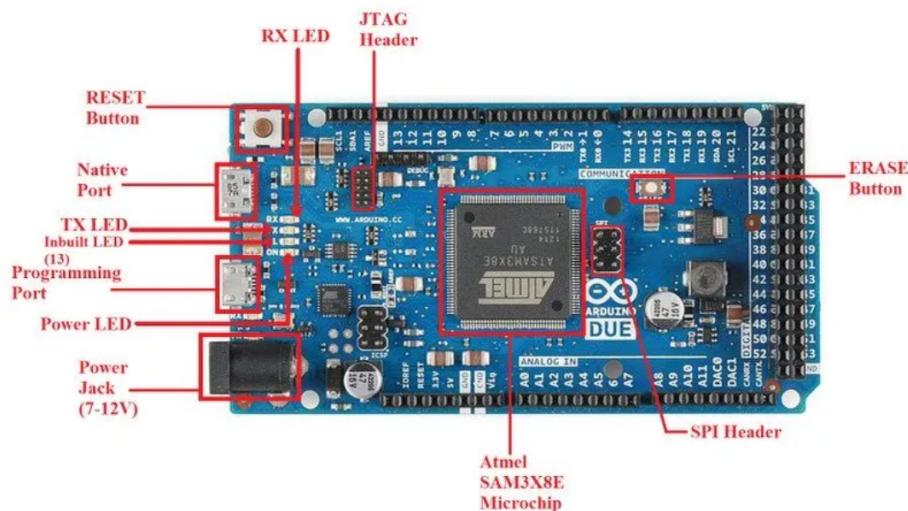


FIGURE 8 – Légende de la carte Arduino Due

Dans l'onglet Outils > Port il faut sélectionner la carte Arduino Due. Ensuite il faut sélectionner le port de programmation comme sur la figure ci-dessous :

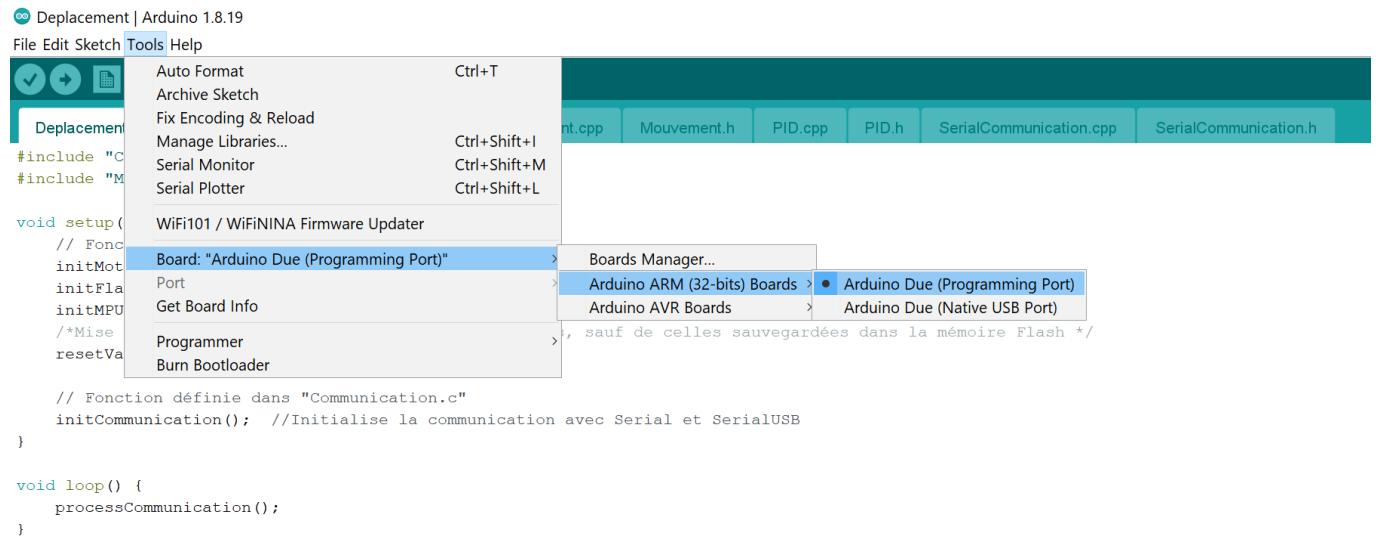


FIGURE 9 – Sélection du port de programmation

Il suffit ensuite d'upload le code avec la flèche =>.

3.3 Prise en main

La partie de prise en main du robot nous a pris beaucoup de temps. Nous avons commencé par récupérer les fichiers du stagiaire Anouar WALZIKI qui à travaillé sur le robot durant son stage de deuxième année à l'ENSEIRB-Matmeca. Le rapport de stage de Anouar WALZIKI sera inclus dans le dossier. Il apport d'avantage de précision sur le fonctionnement du robot. Après la lecture de son rapport de stage, nous avons pu comprendre dans les grandes lignes le fonctionnement global du robot, l'utilité des périphériques externes, la communication avec la carte Arduino Due, ainsi que l'objectif/la mission du robot.

Nous avons ensuite analysé tous les codes différents qui étaient présent dans le répertoire de Anouar. Il existent plusieurs codes qui permettent de réaliser plusieurs scénarios de déplacement du robot en fonction des obstacles et de l'environnement d'évolution. Nous avons testé plusieurs code pour vérifier leur fonctionnement.

La prochaine étape était de comprendre comment fonctionnait le capteur LiDar et la Caméra Intel RealSense. Afin que ces périphériques puissent être utilisés, des bibliothèques spéciales ont du être importées. Les appareils sont connectés à l'ordinateur par l'intermédiaire de ports série.

Pour que le code puisse bien communiquer avec le LiDar, la caméra ainsi que la carte Arduino, il est impératif de définir dans le code sur quel port série (COM) ces derniers sont connectés. Pour le LiDar et la caméra, le port COM associé est toujours fixe. Cependant pour l'Arduino, le port de connexion n'est pas statique. Il faut donc aller dans le Panneau de configuration > Gestion des périphériques > Et vérifier sur quel port COM la carte est connectée avant chaque lancement du robot.

3.4 Fonctionnement et commande de l'Arduino

Le module Arduino permet le contrôle des roues mecanum du robot. En effet il est possible de contrôler chaque moteur de manière indépendante. La fonction `write,read()` permet d'envoyer à la carte Arduino des chaînes de caractère converties en bit sur le port série. Il est préférable d'utiliser le "Native PORT" pour envoyer les commandes car leur fréquence sera élevée. L'ensemble des commandes sont disponibles dans le dossier constructeur, mais les commandes que nous allons principalement utiliser sont les suivantes :

- "B" : Débute la communication avec la base
- "V" : Active/désactive la verbosité du log
- "R" : Remet à leur valeur initiale l'ensemble des variables (sauf celles stockées dans la mémoire Flash)
- "r" : Remet à leur valeur par défaut les variables stockées dans la mémoire Flash (coefficients des Pid et position de départ)
- "M" : Numéro du moteur avec lequel on souhaite communiquer (de 0 à 3 pour piloter un moteur et 4 pour tous les piloter)
- "W" : Envoi un PWM à un ou tous les moteurs suivant celui sélectionné (pwm de -2100 à 2100 envoyé avant la lettre)
- "x/y/z A t" : Envoi un ordre de position cible au robot dans la carte dans le mode de déplacement Automatique (valeurs en mm ou ° renseignées avant la lettre)

On donne maintenant des exemples de cas d'utilisation de ces commandes (hors les commandes "B", "R" et "r" qu'on utilise directement sans modification) :

- "4M0W" : On envoie un PWM de 0 à tout les moteurs, le robot s'arrête.
- "100xAt" : On avance le robot à la position d'abscisse 100mm
- "200yAt" : On avance le robot à la position d'ordonnée 100mm
- "30zAt" : on fait tourner le robot de 30°
- "100xAt/200yat/30zAt" : On effectue toutes les tâches citées avant

4 Objectifs

L'objectif final est centré sur l'amélioration de la programmation de la plate forme mobile. Le but étant d'intégrer le LiDAR et la caméra 360° pour permettre un déplacement autonome du véhicule. Nous avons décidé de mettre en place une interface graphique pour simplifier l'utilisation et le paramétrage du robot, ainsi que la compréhension du code. Nous avons tenté d'améliorer les performances du code et de faire des modifications qui nous semblaient pertinentes comme la modification de la position du LiDar par exemple. Enfin, pour pouvoir avoir une cartographie en 2D à 360°, nous avons décider de créer un support pour l'ordinateur de qui minimiserai son impact sur la fidélité du scan de la pièce fait par le LiDar.

La répartition des différentes tâches de ce projet s'est faite selon le diagramme de Gantt ci-dessous :

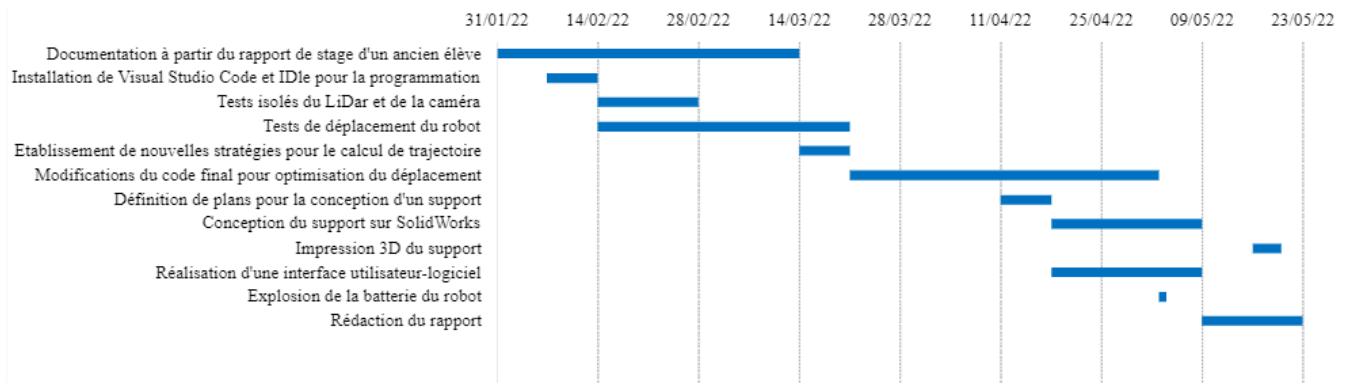


FIGURE 10 – Diagramme de Gantt de l'avancement du projet

5 Améliorations

5.1 Au niveau du code

Le code final mis en annexe permet le déplacement du robot sans s'arrêter. Il est composé de plusieurs fonctions qui permettent de calculer les points d'une nouvelle trajectoire ($update_{path}()$), de mettre à jour la carte en refaisant une nouvelle acquisition du relief avec le capteur LiDAR ($update_{map}()$), d'envoyer les instructions de déplacement à l'Arduino ($write, read()$), de vérifier si un obstacle est proche ($ObstacleProche()$),

Choix de la position d'origine du robot (position initiale du LiDar), ainsi que de vérifier si le robot est bien arrivé à destination ($ObjectifAtteint()$).

C'est fonctions sont appelées dans une boucle while tant que l'objectif n'est pas atteint. Lors du passage dans la boucle, tout d'abord le carte est mise à jour, puis une trajectoire est calculée et ensuite en fonction de si la voie est libre, le robot exécute les déplacements prévus en envoyant les commandes de déplacements à la carte Arduino. Si un obstacle est repéré, le robot recalcule un itinéraire et change de direction.

5.2 Mise en place d'une interface

Nous avons aussi décidé de programmer une interface afin de faciliter le paramétrage du programme ainsi que son lancement. Cette interface a été codée avec la bibliothèque Tkinter. Cette interface présente les entrées suivantes :

- La position X et Y du point d'arrivée
- L'angle d'orientation du robot à la fin

- Le rayon de sécurité du robot
- L'option d'afficher la carte 2D
- D'autoriser le déplacement du robot
- De paramétrier le port série de la carte Arduino

Il est possible de lancer un scan rapide de la pièce avant de lancer le déplacement du robot afin de s'assurer de la bonne acquisition de l'environnement.

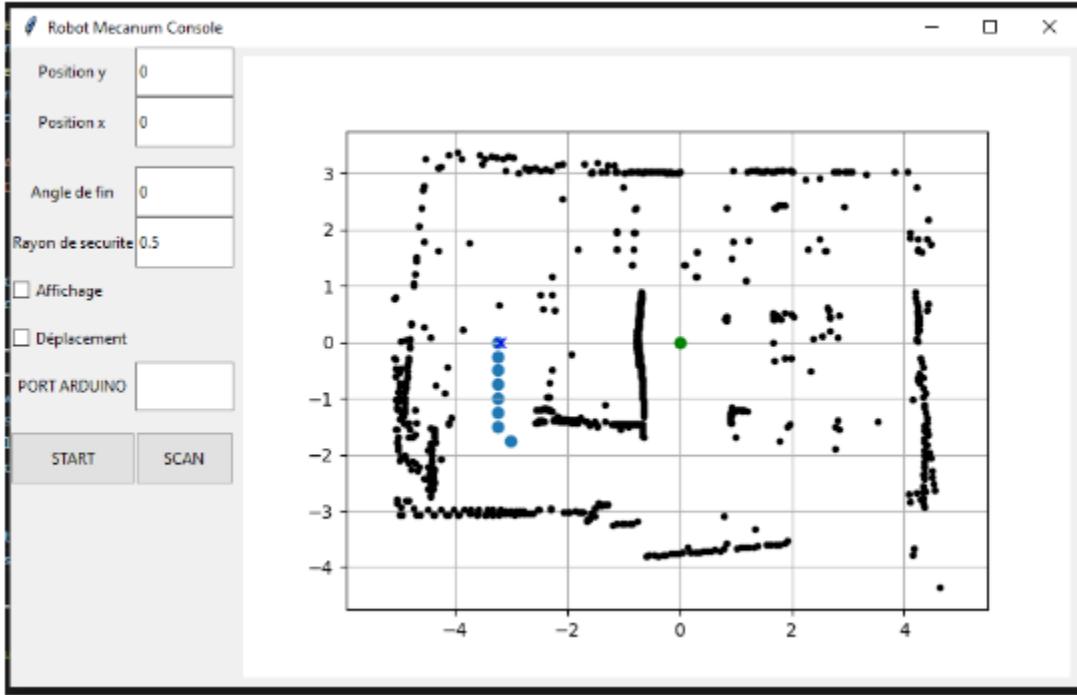


FIGURE 11 – Visuel de l'interface utilisateur

5.3 Fabrication d'un support

Nous avons constaté au cours de nos tests que la position du LiDar sur la plateforme pouvait amener à quelques soucis. En effet, ne disposant pas de carte modèle Raspberry Pi qui aurait permis un contrôle à distance du robot sans l'utilisation d'un ordinateur encombrant, nous étions contraints d'embarquer sur le robot notre PC sur lequel nous exécutions les codes pour le déplacement. La méthode initialement choisie a été de poser l'ordinateur sur un petit support à 4 pieds, lui-même posé sur la plate-forme. Le LiDar, lui, était placé juste en-dessous de ce petit support.



FIGURE 12 – Utilisation initiale du robot avec PC posé sur le petit support

Néanmoins, cela a entraîné la détection des pieds de celui-ci lors du scan et par conséquent, une cartographie en 2D erronée. Nous pouvions constater ce problème sur la map, comme 4 points situés tout autour du point d'origine, correspondant à la position du robot dans l'environnement. L'algorithme de déplacement considérait qu'il y avait une ouverture dans le mur qui se trouvait derrière le pied du support. En solution, nous avons donc choisi de retirer ce petit support et de placer notre PC directement sur la plate-forme. Celui-ci était donc placé juste derrière le LiDar. Cela a posé un nouveau problème lors du scan du LiDar qui détectait donc un obstacle juste derrière lui, générant ainsi des incohérences pour l'algorithme de trajectoire qui effectuait des rotation sur lui même.

De plus, la cartographie de l'espace en 2D n'était pas faite à 360°. Une solution pour l'algorithme de trajectoire aurait pu se trouver derrière le robot sans qu'il le sache. Il ne pouvait donc pas reculer. Pour palier ce problème, nous avons donc imaginer comme solution de surélever le PC de sorte que le LiDar ne le détecte pas pendant son scan à 360°.

Pour ce faire, nous avons conçu une pièce, sur le logiciel de CAO SolidWorks, qui servirait de support pour le PC.

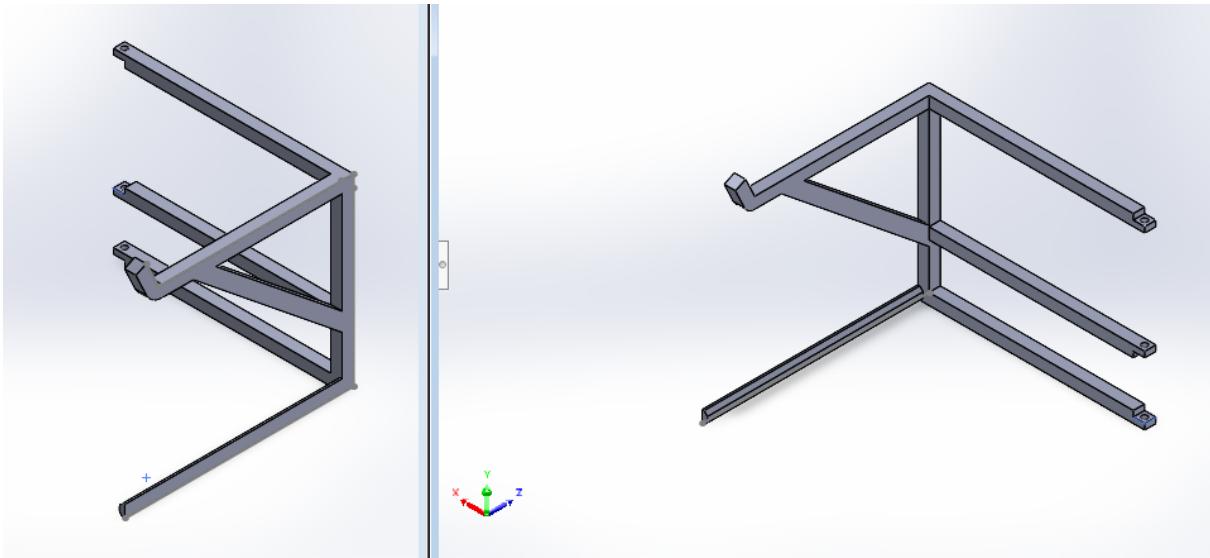


FIGURE 13 – Plans SolidWorks des 2 pièces du support

Les mesures ont préalablement été effectuées sur la structure du robot avant d'être reportées sur le logiciel. La structure du robot, sur laquelle est reposée la plateforme, étant des rails, nous avons profité de cette caractéristique pour concevoir les pieds du support avec un profil adapté pour venir se glisser dans ces dits rails. Nous avons ensuite imprimé cette pièce grâce à l'imprimante 3D du FabLAB de l'école. Il fallait en revanche faire attention aux dimensions de celle-ci. En effet, la largeur du robot était plus grande que celle de l'imprimante. Nous avons donc du concevoir le support en 2 parties complémentaires que nous avons ensuite fixées ensemble par vis-écrou. Ainsi, la contrainte de marche arrière est révolue.

Évidemment cette solution ne peut fonctionner que pour une configuration sans le bras robot qui peut se fixer sur la plate-forme dans une deuxième utilisation. Dans ce cas de figure, l'utilisation d'une carte Raspberry est nécessaire.

6 Problèmes rencontrés

6.1 Câblage des sorties de la carte Arduino

Nous avons rencontré un problème de faux contact durant notre travail. En effet, le comportement individuel des roues du robot ne correspondait pas aux commandes envoyées à l'Arduino. Pour une simple commande d'avancement rectiligne, certaines roues tournaient dans le sens inverse et l'une des roues tournait à sa vitesse maximale. Si ce problème se reproduit, il ne faut pas hésiter à vérifier tous les branchements de la carte Arduino ainsi que les servomoteurs. Il serait même judicieux de souder les fils une fois que le robot est opérationnel.

6.2 Incendie de la batterie Lipo

Un incident, heureusement sans conséquences graves, s'est produit pendant notre projet. En effet, la batterie LiPo, qui alimente le robot, a pris feu le 3 mai. Il s'agissait en fait d'un emballage thermique dû à une mauvaise manipulation lors de la recharge de celle-ci. Nous avions récupéré, au début du projet, le robot présentant une batterie déjà défectueuse car nous ne parvenions pas à la recharger via le chargeur adapté initialement prévu à cet effet. Étant contraint de recharger le robot avant chaque nouvelle séance de projet, la solution adoptée a été celle de brancher une alimentation de tension aux bornes prévues pour le relevé de tension. Cette méthode est très déconseillé car les cellules qui composent la batterie doivent être chargées de manière équilibrée. En effet, chaque cellule ne présentant pas la même tension à ses bornes, il y en a qui sont plus déchargées que d'autres et nécessitent donc une recharge plus adaptée. En utilisant un générateur de tension, l'équilibre entre les cellules de sont pas respecté, ce qui conduit à des dégradations de la batterie. Enfin, comme les cellules présentaient une très grande sous-tension (2,4V par cellule), l'état de ces dernières s'est encore dégradé ce qui a inévitablement conduit à une instabilité des cellules jusqu'à un emballage thermique caractérisé par une formation de gaz inflammable avec une très forte température, ainsi que de la fumée toxique.

Les consignes de sécurité comme l'utilisation d'une poche ignifugé autour de la batterie, ont permises de contenir grandement les flammes ainsi que la température.

Normalement la prochaine version du robot devrait intégrer un BMS (Battery Management System). C'est un circuit qui s'occupe de gérer la recharge et la décharge de la batterie LiPo. Il est aussi conseillé d'utiliser un bipper/buzzer qui permet d'alerter lorsque la batterie a un problème.

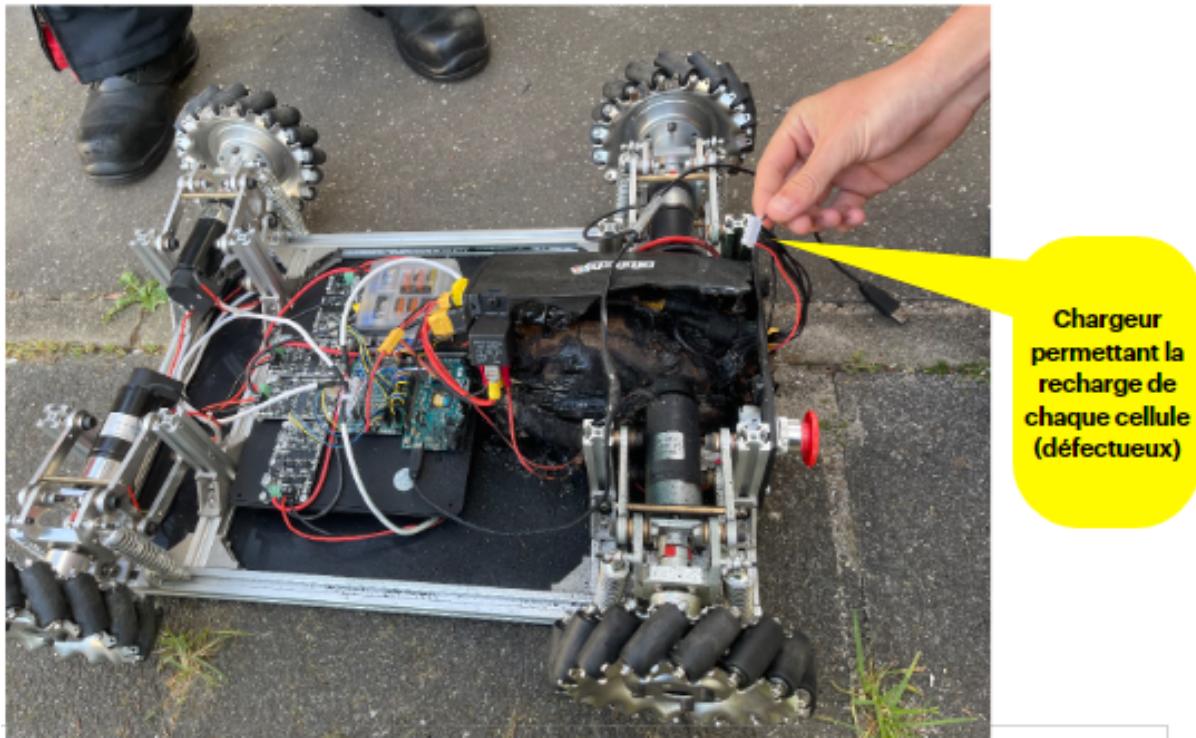


FIGURE 14 – Photo du robot après explosion de la batterie

De plus, cet incident est tombé au mauvais moment, car nous venions, la séance précédente, de débugger le problème de câblage qui nous empêchait de tester sur le robot l'amélioration du code décrit précédemment. Plus précisément, après avoir résolu ce dit problème, nous avons effectué un test d'utilisation, et c'est à ce moment précis que le robot est tombé à cours de batterie. Les dommages constatés sur le robot ont été les suivants :

- la batterie LiPo
- les 2 moteurs contrôlant les roues arrières
- La plateforme

Par chance, le reste du montage a été épargné. Les circuits de la partie puissance, composée des drivers moteurs et du porte fusible, ainsi que la partie commande, composée de la carte Arduino, de la centrale inertielles et du convertisseur USB-UART, ont tous pu être récupérés. Le robot a ensuite été renvoyé à Robot-Maker pour réparations.

7 Pistes d'améliorables futures

Il y a plusieurs pistes d'amélioration possible avec ce projet. Nous pouvons en citer quelques unes : Il serait intéressant de recalculer une nouvelle carte 2D au moins toutes les secondes, de recalculer une nouvelle trajectoire à la même fréquence et en ajustant constamment la position du point d'arrivée en fonction du déplacement déjà effectué. Ceci consisterai à avoir une carte de la pièce dynamique avec comme point centrale le robot lui même. Ceci pourrait permettre de mieux esquiver les obstacles et toujours avoir une carte à jour.

Il est aussi possible d'implémenter le code de déplacement sur une Raspberry Pi afin de se débarrasser de l'encombrement d'un PC sur le robot.

8 Conclusion

Finalement, malgré quelques incidents rencontrés au cours du projet tels que l'explosion de la batterie ou le problème de câblage, nous sommes tout de même parvenus à cocher 3 cases sur 4 du cahier des charges.

Parmi les différents codes que nous avions reçus d'Anouar, nous sommes parvenus à étudier et comprendre ce que chacun d'eux permettait de faire, que ce soit le simple contrôle du LiDar pour la cartographie de l'environnement ou bien le déplacement effectif du robot.

Cependant, nous nous sommes bien rendus compte de la difficulté que représente la relecture par une personne B d'un code déjà bien complet et complexe réalisé par une personne A. C'est pourquoi la réalisation de l'interface utilisateur était nécessaire afin d'utiliser les codes sans avoir à se plonger directement dedans et passer du temps à comprendre que réalise chaque ligne (sauf si le projet avait été basé sur de la programmation pure évidemment). Grâce à cette interface, un éventuel prochain utilisateur pourra plus facilement prendre en main le robot.

De plus, grâce à la conception du support, nous avons pu retirer une contrainte de déplacement du robot qui était celle de ne pas pouvoir reculer. En retirant le PC de derrière le LiDar, ce dernier peut alors effectuer un véritable scan à 360° sans détecter un obstacle (de type mur) juste derrière lui. En rentrant comme coordonnées d'arrivée un point qui se trouve à derrière lui, le robot pourra désormais tracer une trajectoire cohérente et l'atteindre en effectuant une rotation.

Enfin, après avoir résolu le problème de câblage, nous étions en mesure de tester les modifications que nous avions effectuées sur le code pour améliorer son déplacement comme nous le souhaitions. Mais c'était sans compter sur la décharge totale du robot en cours d'utilisation qui nous a forcé à devoir le mettre à la charge.

Nous espérons que le prochaine élève/groupe qui travaillera sur ce robot, une fois revenu de réparation, sera en mesure d'y appliquer les modifications que nous allons lui laisser pour les tester sur un robot fonctionnel.

9 Annexes

1 Caractéristiques techniques

1.1 RPLiDar A2M8

Tracking Module Properties

Imaging Module	Intel® RealSense™ Tracking Module T261
Baseline (mm)	64±0.15
Left/Right Fisheye Imagers	DV9282
Shutter Type	Global
Fisheye FOV (degrees)	D:173
Module Dimensions (mm)	X=93.35 (+0.15 -0.25) Y=17.60±0.15 Z=7.13±0.30

Inertial Measurement Specifications

Parameter	Properties
Degrees of Freedom	6
Acceleration Range	±4g
Accelerometer Sample Rate	62.5Hz
Gyroscope Range	±2000 Deg/s
Gyroscope Sample Rate	200Hz

1.2 Caméra Realsense T265

- For Model A2M7/A2M8 Only

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter[m]	0.15	-	8	Based on white objects with 70% reflectivity
Angular Range	Degree	-	0-360	-	-
Distance Resolution	mm	-	<0.5 <1% of the distance	-	<1.5 meters All distance range*
Angular Resolution	Degree	0.45	0.9	1.35	10Hz scan rate
Sample Duration	Millisecond(ms)	-	0.25	-	-
Sample Frequency	Hz	2000	4000	4100	The rate is for a round of scan. The typical value is measured when RPLIDAR takes 400 samples per scan
Scan Rate	Hz	5	10	15	The rate is for a round of scan. The typical value is measured when RPLIDAR takes 400 samples per scan

1.3 Code de la fonction main() du fichier rcup_finale.py

```
444 def main():
445     root = tk.Tk()
446     root.title("Robot Mecanum Console")
447
448     ### ----- Dimension de la fenetre et centrage ----- ###
449
450     window_width = 1000
451     window_height = 600
452
453     # get the screen dimension
454     screen_width = root.winfo_screenwidth()
455     screen_height = root.winfo_screenheight()
456
457     # find the center point
458     center_x = int(screen_width/2 - window_width / 2)
459     center_y = int(screen_height/2 - window_height / 2)
460
461     # set the position of the window to the center of the screen
462     #root.geometry(f'{window_width}x{window_height}+{center_x}+{center_y}')
463
464     move_robot = tk.BooleanVar()
465     show_animation = tk.BooleanVar()
466
467     ly = ttk.Label(root, text="Position y")
468     lx = ttk.Label(root, text="Position x")
469     langle = ttk.Label(root, text="Angle de fin")
470     lrayon = ttk.Label(root, text="Rayon de securite")
471     lport = ttk.Label(root, text="PORT ARDUINO")
472
473     Entry_gy = ttk.Entry(root, width=5)
474     Entry_gy.insert(0, "0")
475     Entry_gx = ttk.Entry(root, width=5)
476     Entry_gx.insert(0, "0")
477     Entry_angle_orientation_finale = ttk.Entry(root, width=5)
478     Entry_angle_orientation_finale.insert(0, "0")
479     Entry_rayon = ttk.Entry(root, width=5)
480     Entry_rayon.insert(0, "0.5")
481     port_arduino = ttk.Entry(root, width=7)
482
483     Checkbutton_deplacement = ttk.Checkbutton(root, text="Déplacement", variable=move_robot, onvalue=True, offvalue=False)
484     Checkbutton_affichage = ttk.Checkbutton(root, text="Affichage", variable=show_animation, onvalue=True, offvalue=False)
485
486
487     img = tk.PhotoImage(file='C:/Users/Loys Forget/Documents/VSCode/RCUP_Mecanum/RCUP_Mecanum/Code complet/Mecanum/2D_map/figure.png')
488     img1 = img.subsample(1, 1)
489
```

```

489
490     def quick_scan():
491         pipe = cm.initcam()
492         ox, oy = [], []
493         tmp1 = []
494         tmp1 = update_map(pipe)
495         ox.extend(tmp1[0])
496         oy.extend(tmp1[1])
497         plt.plot(ox, oy, ".k")
498         plt.grid(True)
499         plt.axis("equal")
500         plt.savefig('C:/Users/Loys Forget/Documents/VSCode/RCUP_Mecanum/RCUP_Mecanum/Code_complet/Mecanum/2D_map/figure.png')
501
502     def button_run():
503         gy = float(Entry_gy.get())
504         gx = float(Entry_gx.get())
505         robot_radius = float(Entry_rayon.get())
506         angle_orientation_finale = float(Entry_angle_orientation_finale.get())
507         ArduinoSerialPort = port_arduino.get()
508         print(gy, gx, " ", robot_radius, angle_orientation_finale, ArduinoSerialPort, move_robot, show_animation)
509         run(grid_size=0.25, robot_radius=robot_radius, show_animation=show_animation, move_robot=move_robot, ArduinoSerialPort=ArduinoSerialPort, gx=gx, gy=gy, angle_orientation_finale=angle_orientation_finale)
510
511     Button_run = ttk.Button(root, text="START", command=button_run)
512     Button_scan = ttk.Button(root, text="SCAN", command=quick_scan)
513
514     ly.grid(row=0, column=0, pady=2)
515     lx.grid(row=1, column=0, pady=2)
516     Entry_gy.grid(row=0, column=1, stick='nsew')
517     Entry_gx.grid(row=1, column=1, stick='nsew')
518
519     langle.grid(row=3, column=0, pady=2)
520     lrayon.grid(row=4, column=0, pady=2)
521     Entry_angle_orientation_finale.grid(row=3, column=1, stick='nsew')
522     Entry_rayon.grid(row=4, column=1, stick='nsew')
523     Checkbutton_affichage.grid(row=5, column=0, stick='nsew')
524     Checkbutton_deplacement.grid(row=6, column=0, stick='nsew')
525     lport.grid(row=7, column=0, pady=2)
526     port_arduino.grid(row=7, column=1, stick='nsew')
527     Button_run.grid(row=9, column=0, stick='nsew')
528     Button_scan.grid(row=9, column=1, stick='nsew')
529
530     tk.Label(root, image = img1).grid(row=0, column=3, columnspan=9, rowspan=20, padx=5, pady=5)
531
532     #rcup_finale.parametrage(0.25, 0.5, True, True, 'COM12', -3.2, 0, 0)
533
534     root.mainloop()

```

1.4 Code de la fonction run() du fichier rcup_finale.py

```
286 def run(grid_size, robot_radius, show_animation, move_robot, ArduinoSerialPort, gx, gy, angle_orientation_finale):
287     print(__file__ + " start !!!")
288
289     #POSITION DU LIDAR :
290     lidar_avant = False
291 #----Bloc 1-----
292     pipe = cm.initcam()
293
294     if move_robot:
295         arduino = serial.Serial(port=ArduinoSerialPort, baudrate=115200, timeout=.1)
296         write_read("#B",arduino)
297         write_read("V",arduino)
298         write_read("R",arduino)
299         write_read("r",arduino)
300         angle_orientation = 0
301         rx_old = 0
302         ry_old = 0
303         sx = 0
304         sy = 0
305         xr = 0
306         yr = 0
307         flag = 0
308         Obstacle = False
309         ox, oy = [], []
310
311
312     while(True):
313 #----Bloc 2-----
314         """
315             On vérifie toujours au début si on est arrivé à destination et on arrête le programme si c'est le cas
316             """
317         if(not(ObjectifAtteint(xr, yr, gx, gy))):
318
319             tmp1 = []
320             tmp1 = update_map(pipe, angle_orientation)
321             ox.extend(tmp1[0])
322             oy.extend(tmp1[1])
323
324             if show_animation: # pragma: no cover
325                 plt.plot(ox, oy, ".k")
326                 plt.plot(sx, sy, "og")
327                 plt.plot(gx, gy, "xb")
328                 plt.grid(True)
329                 plt.axis("equal")
330
331         
```

```

331     |
332     |     if lidar_avant:
333     |     |     tmp2 = update_path(ox, oy, grid_size, robot_radius, xr, yr-0.020, gx, gy, show_animation)
334     |     |
335     |     |     else:
336     |     |     |     tmp2 = update_path(ox, oy, grid_size, robot_radius, xr, yr, gx, gy, show_animation)
337     |     |
338     |     rx, ry = [], []
339     |     rx, ry = tmp2[0], tmp2[1]
340
341     |     if show_animation: # pragma: no cover
342     |     |     plt.plot(rx, ry, "-r")
343     |     |     plt.scatter(rx, ry)
344     |     |     plt.savefig('C:/Users/Loys Forget/Documents/VSCode/RCUP_Mecanum/RCUP_Mecanum/Code complet/Mecanum/2D_map/figure.png')
345     |     |     plt.pause(0.001)
346     |     |     plt.draw()
347     |     |     plt.pause(1)
348     |     |     plt.clf()
349
350 #-----Bloc 3-----
351     if move_robot:
352
353         #-----Bloc 3.1-----
354
355         #envoie de la commande pour déplacer le robot suivant le chemin trouvé
356         j=0
357         for j in range(len(rx)):
358             rx_new = rx[len(rx) - j - 1]*1000
359             ry_new = ry[len(rx) - j - 1]*1000
360
361             angle_orientation_old = angle_orientation
362
363             if ((rx_new - rx_old == 0 or abs(rx_new - rx_old) > 0) and ry_new - ry_old > 0):
364                 angle_orientation = 0
365             elif(rx_new - rx_old > 0 and ry_new - ry_old == 0):
366                 angle_orientation = -90
367             elif((rx_new - rx_old < 0 and ry_new - ry_old == 0)):
368                 angle_orientation = 90
369             elif((rx_new - rx_old == 0 or rx_new - rx_old > 0) and ry_new - ry_old < 0):
370                 angle_orientation = -180
371             elif(rx_new - rx_old < 0 and ry_new - ry_old < 0):
372                 angle_orientation = 180
373

```

```

373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424

    ...
    Ajouter une fonction qui arrete le robot si un obstacle est proche
    dans ce cas on reexecute la boucle while
    ...
#----Bloc 3.2-----
Obstacle_=ObstacleProche()

if(Obstacle):
    if(flag==0):
        #faire 4 pas avant de reverifier s'il y a un nouveau obstacle
        print("obstacle proche !!!")
        #write_read("4M0W", arduino)
        flag = 1
        break
    elif(flag==1):
        print("flag == 1")
        flag = 2
    elif(flag==2):
        print("flag == 2")
        flag = 3
    elif(flag==3):
        print("flag == 3")
        flag = 4
    elif(flag==4):
        print("flag == 4")
        flag = 0

#----Bloc 3.3-----
angle_orientation_str = str(angle_orientation)  #---_tmp

if(angle_orientation - angle_orientation_old != 0):
    write_read(angle_orientation_str + "zAt", arduino)
    print("Rotation: " + angle_orientation_str + "zAt")

if(angle_orientation - angle_orientation_old != 0):
    if(abs(angle_orientation - angle_orientation_old) >= 180 and (abs(angle_orientation - angle_orientation_old) != 360)):
        time.sleep(5.5)
        break
    else:
        time.sleep(3.5)
        #break
    rx_old = rx_new
    ry_old = ry_new

    rx_c = str(rx_new)
    ry_c = str(ry_new)

    write_read(rx_c + "xAt/" + ry_c + "yAt", arduino)
    print("Translation : " + rx_c + "xAt/" + ry_c + "yAt")

    xr = rx_old*0.001
    yr = ry_old*0.001

#----Bloc 4-----
else:
    break

angle_orientation_finale_str = str(angle_orientation_finale)
write_read(angle_orientation_finale_str + "zAt", arduino)
print("Rotation finale: " + angle_orientation_finale_str + "zAt")

print("")
print("Objectif atteint !!!")

pipe.stop()

pass

```