Rapport technique

Marie DURIVAUX - Colin GUIRARDEL - Gabriel PRADAT-PEYRE - Rosalie TABARIE

6 avril 2023

Ce document s'adresse à quiconque souhaitant reprendre la programmation du robot `0-113` et de son interface de contrôle à distance. Ci-dessous sont décrits et expliqués le projet, son cahier des charges ainsi que tout ce qui a déjà réalisé par notre équipe ainsi que les points qu'il reste à terminer ou améliorer.

Table des matières

1 Introduction

1.1 Bref historique du projet

SOLEC est venu vers nous le 23 janvier 2023 avec la proposition d'un projet de réalisation d'un robot commandé à distance et capable de se déplacer dans un entrepôt dans le cadre de notre formation d'élève ingénieur à l'Institut d'Optique. C'est bien sûr avec grand enthousiasme que nous nous y sommes attaqués.

Ce projet a pris fin le mercredi 29/03. Ce document décrit l'état du prototype à cette date.

1.1.1 Calendrier

- Séance 1 - 25/01 - Début du projet, planning initial
- Séance 3 - 01/03 - Point d'équipe sur l'avancement, réévaluation du planning
- Séance 5 - 15/03 - Audit tournant
- Séance 7 - 22/03 - Réunion des parties trajet et mécatronique, fin du projet
- Séance 8 - 29/03 - Audit du projet par SOLEC

1.2 Objectif du robot et présentation

Le projet consiste a permettre à un robot de se déplacer sans capteurs ni balises dans un entrepôt dont le plan est connu à l'avance. Les consignes seront envoyées depuis un ordinateur *via* une interface graphique.

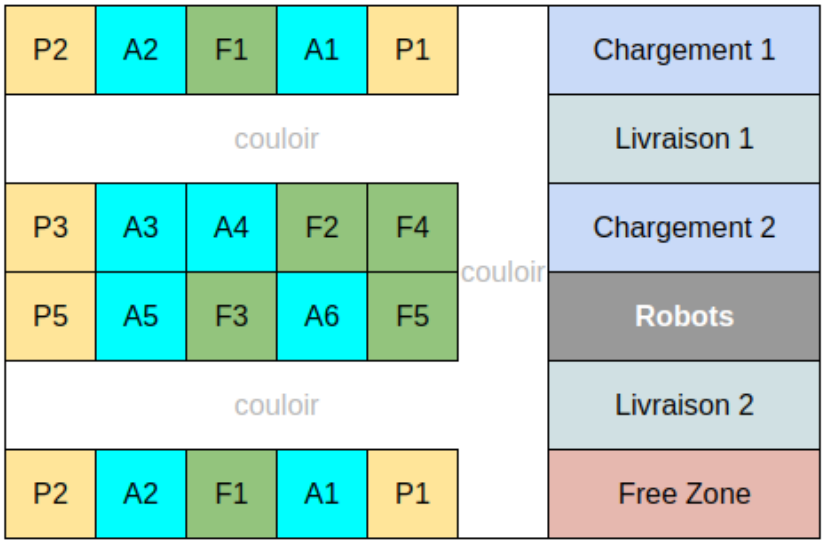

FIGURE 1.1 – Plan de l'entrepôt SOLEC type - Source SOLEC

Le châssis du robot a été fourni par le LEnsE.

1.3 Notice d'utilisation

Le prototype n'est pas terminé, en particulier l'interface humain-machine n'a pas été reliée au prototype.

1.3.1 Interface humain-machine

L'interface humain-machine est lancée en exécutant le programme `colin_interface_s1.py`.

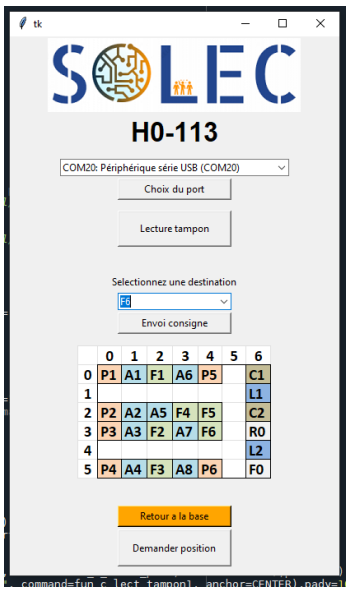

FIGURE 1.2 – Cahier des charge et performances

La première section de l'interface sert à gérer la liaison série entre l'ordinateur et la carte. Avant toute chose, il faut sélectionner cette liaison série dans le menu déroulant, puis appuyer sur `Choix du port`. Ceci initialisera la liaison série. Un message devrait apparaître dans la console Python.
Le bouton `Lecture tampon` est un outil de débugage qui permet de lire l'intégralité du tampon de la liaison série entre la carte Nucléo et l'ordinateur.

La seconde section permet d'envoyer des consignes de déplacement au robot. Un menu déroulant permet de choisir la consigne à envoyer, et le bouton `Envoi consigne` permet de demander au robot de s'y rendre.

En dessous du plan, les boutons `Retour à la base` et `Demander position` permettent d'envoyer respectivement les consignes `'R0'` (qui demande au robot de se rendre en R0) et `'Q0'` (qui a terme devrait demander au robot d'indiquer sa position, mais n'est pas implémentée).

À terme, un bouton `Initialisation` devrait indiquer au robot qu'il est dans la zone de départ R0.

1.3.2 Communication Bluetooth

Le robot porte un module Bluetooth `NRF24`. cf plus loin pour branchements

Une autre carte Arduino + TUNIS porte un second module Bluetooth. Cette seconde carte doit être connectée à un ordinateur *via* une liaison USB. Il est impossible d'utiliser `TeraTerm` pour lire la liaison série de cette carte car le port série correspondant devra être utilisé par `Python` pour l'interface graphique.

Le programme `colin_IeTI_nRF24_transfer_data` devra être chargé dans cette carte. L'adresse utilisée pour ce module est `0xE7E7E7E7E6`. Elle peut être changée dans le fichier `MOD24_RNF.h`. L'adresse utilisée doit être la même pour les deux modules, et différente de celles utilisées par les modules d'autres projets travaillant à proximité ($\sim$ 50 m).

Pour le moment, une troisième carte sur laquelle le module Bluetooth du robot peut être installé est utilisée pour les tests Bluetooth. Il convient donc de la brancher elle aussi à la liaison série d'un ordinateur.
On pourra lire les informations sur un logiciel comme `TeraTerm`.

1.3.3 Prototype

En attendant qu'il soit relié à l'interface, le robot a été programmé pour que les tests se déclenchent à l'aide du bouton poussoir présent sur la carte nucléo. Pour réaliser cela, il faut définir dans l'en-tête du programme l'entrée correspondant au bouton avec l' instruction :

```
//Bouton poussoir de la carte nucléo : sert pour tester l'interruption
    par une consigne
InterruptIn mon_int(PC_13);
bool etat=0; //Initialisation de l'interrupteur
```

On met ensuite l'instruction suivante dans le *main* du programme :

```
mon_int.rise(&FONCTION_INTERRUPTION);
```

FONCTION_INTERRUPTION est ici une fonction d'interruption standard qu'on peut utiliser pour exécuter la fonction qu'on souhaite tester en appuyant sur le bouton. La syntaxe de *FONCTION_INTERRUPTION* est alors la suivante :

```
void FONCTION_INTERRUPTION()
{

    // ACTIONS A REALISER LORS DE L'ACTIVATION DE L'INTERRUPTION 1

        etat = 1-etat; // changement d'etat allumé/éteint (détection
            front montant bouton)

        if(etat == 1){

         FONCTION_TESTEE();

        }
        else{

            //Moteur 1
            fun_RG_stop_mot1();


            //Moteur 2
            fun_RG_stop_mot2();


            //Moteur 3
            fun_RG_stop_mot3();
        }
}
```

En appuyant une première fois sur le bouton on exécute la fonction qu'on veut tester, et on stoppe le robot en rappuyant dessus.

1.3.4 Nomenclature

Noms de variables La programmation embarquée implique de travailler avec de nombreuses variables globales. Afin d'éviter les conflits de noms, et de trouver plus facilement l'usage d'une variable, il a été mis en place le procédé suivant : le nom des variables doit contenir `fun` pour une fonction et `tab` pour un tableau, ainsi que l'initiale de son créateur. Ainsi, `fun_RG_mot1` est une fonction écrite par Rosalie et Gabriel (bloc 'mécatronique') et `tab_c_data` est un tableau défini par Colin (bloc 'interface & communication'). Le reste du nom est sensé être explicite.

1.4 Cahier des charges et bilan des performances

Nom	Critère	Valeur	Validation ?
Rapidité	Vitesse comprise entre 10 et 30 cm/s	~10,3 cm/s	VALIDE
Autonomie	Parcours de 1km sans recharge nécessaire	N/A	N/A
Fiabilité en position	Erreur maximale de 2 cm sur 1 m	90cm ± 2mm	VALIDE
Fiabilité en angle horaire	Erreur maximale de 3° sur 360°	360° + 3°	LIMITE
Fiabilité en angle trigonométrique	Erreur maximale de 3° sur 360°	360° - 4,8°	INVALIDE
Ergonomie	IHM accessible sans formation	OK	VALIDE
Collection des donées	Temps réel	Non implémenté	INVALIDE

FIGURE 1.3 – Cahier des charge et performances

1.5 Schéma fonctionnel

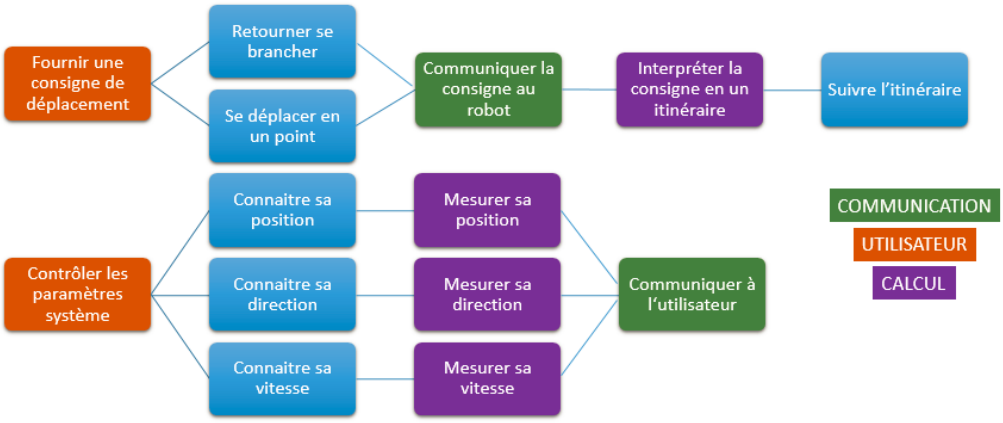

FIGURE 1.4 – Schéma fonctionnel

2 Bloc "trajet"

Cette partie correspond à la conception et le développement de l'algorithme calculant le trajet du robot entre un point de départ et l'arrivée. Cette partie, qui nécessite de traduire la commande de l'utilisateur en une suite de commande compréhensibles par le robot est intégralement traitée sur la carte nucléo du robot.

2.1 Définition du pavage

A partir du plan de l'entrepôt fourni par SOLEC, nous avons défini un pavage de cet entrepôt assignant à chaque case une position unique.

	0	1	2	3	4	5	6
0	P1	A1	F1	A6	P5		C1
1							L1
2	P2	A2	A5	F4	F5		C2
3	P3	A3	F2	A7	F6		R0
4							L2
5	P4	A4	F3	A8	P6		F0

FIGURE 2.1 – Plan de l'entrepôt considéré

	0	1	2	3	4	5	6
0	1	2	3	2	1	0	4
1	0	0	0	0	0	0	6
2	1	2	2	3	3	0	4
3	1	2	3	2	3	0	5
4	0	0	0	0	0	0	6
5	1	2	3	2	1	0	7

FIGURE 2.2 – Matrice du plan de l'entrepôt

Les cases de couloir se voient assignées une valeur binaire nulle et toutes les autres cases une valeur non nulle correspondant à leur type de case. Ainsi le robot navigue dans les couloirs (cases nulles) et n'entre dans un bloc (case non nulle) que sur demande explicite - c'est-à-dire lorsqu'il s'agit de la case d'arrivée demandée par l'utilisateur. Dans l'éventualité où le robot serait amené à optimiser de lui-même un itinéraire passant par plusieurs type de case, on pourra mettre à profit la codification utilisée.

2.2 Fonctionnement général de l'algorithme

L'algorithme calcule le chemin case par case. A chaque case, le robot cherche à se rapprocher de sa destination tout en empruntant uniquement les cases couloirs. Un tel algorithme est aisé à coder, mais reste bloqué lorsqu'il s'agit de passer de la partie haute de l'entrepôt à la partie basse (par exemple, aller de A6 en P4). Pour pallier à ce problème tout en conservant une structure d'algorithme simple, une case destination intermédiaire est définie dans ces cas pour obliger le robot à passer par le couloir vertical de droite faisant communiquer les deux parties de l'entrepôt. Dans le cas d'un changement du plan de l'entrepôt ou d'un agrandissement de celui-ci, il pourrait être pertinent de remplacer cet algorithme par un solveur de labyrinthe standard qui peut être trouvé sur Internet, ou codé à la main. Il est probable en revanche que cela augmente le temps de calcul.
Cette étape de définition case par case du chemin fournit en sortie la liste des positions successives que le robot doit emprunter afin d'atteindre sa destination.

2.3 Complexité

L'algorithme actuel effectue un nombre fixe d'opération par case de déplacement demandé, sa complexité est donc proportionnelle à la longueur du trajet. Sachant que pour l'entrepôt considéré, le nombre de déplacement entre deux positions est au maximum de 15, cette complexité est tout à fait raisonnable pour notre carte.

2.4 Passage des coordonnées des cases aux instructions robot

Une fois le chemin calculé case par case, une deuxième fonction traduit la suite de position en une suite d'actions que le robot doit effectuer. Cette suite d'action dépends de deux paramtères : la position du robot dans l'entrepôt et sa direction. Par exemple, si le robot se situe en case (ligne, colonne)=(4,2) tourné vers le Nord et doit se déplacer en case (4,3), le robot devra effectuer une rotation dans le sens horaire et avancer d'une case. Cette suite d'action est codifiée sous la forme d'un string de longueur inconnue (donc défini avec une taille de 100) dont les lettres correspondent à une action :

Lettre	Fonction
A	Avancer d'une case
H	Tourner de 90° dans le sens Horaire
T	Tourner de 90° dans le sens Trigonométrique
S	Marque la fin d'une instruction et arrête le robot

2.5 Exemple

Supposons que le robot démarre en case R0=(3,6) orienté vers le Nord avec la consigne utilisateur de se rendre en A7=(3,3).

— 1. Détermination des cases vides adjacentes aux consignes : arrivée en (4,3)
— 2. Calcul du chemin case par case. Dans notre cas, il n'est pas nécéssaire de définir une destination intermédiaire. Résultat : [(3,5), (4,5), (4,4), (4,3), (3,3)]
— 3. Traduction du chemin en actions. Le robot commence orienté vers le Nord, il doit donc d'abord se tourner vers l'Ouest afin de pouvoir avancer tout droit.
Résultat : "TATAHAAHAS"

3 Bloc mécatronique

Dans cette partie, nous décrivons les fonctions qui pilotent directement le robot. Elles sont traitées par la carte nucléo du robot.

3.1 Faire tourner les roues-Fonctions de bases

Le robot est présenté ci-dessous avec la localisation de ses trois moteurs ainsi que la direction dans laquelle on le fait avancer :

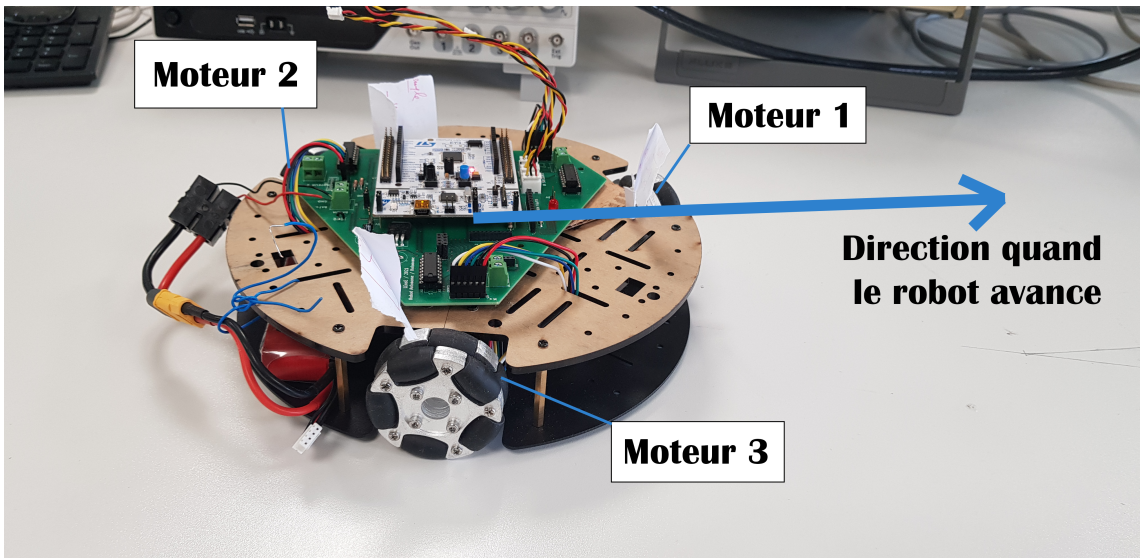

FIGURE 3.1 – Photo du robot et indication de ses moteurs et de sa direction de déplacement

Pour que le robot puisse avancer, il faut donner l'ordre à ses moteurs de faire tourner les roues. Les moteurs utilisés fonctionnent avec un courant continu, on va les commander à l'aide d'une sortie PWM. On initialise la période à 10 ms. Chaque moteur possède deux entrées, notées sur la documentation. En fonction de sur quelle entrée on envoie le rapport cyclique, le moteur tournera dans le sens horaire ou dans le sens trigonométrique. On créé des variables `RG_Sens_rota1, RG_Sens_rota2, RG_Sens_rota3` a qui on donnera la valeur 1 si on veut faire tourner le moteur dans le sens trigonométrique et -1 si on veut le faire tourner dans le sens horaire. On a créé 3 fonctions qui font tourner les moteurs dans le sens voulu avec le rapport cyclique voulu. Elles serviront de briques de bases aux fonctions plus complexes.

3.2 Donner des commandes simples

A partir des fonctions de base, nous avons construit des commandes simples pour faire avancer le robot et le faire tourner. Nous avons choisi de programmer le robot pour qu'il parcourt exactement une case du pavage (défini dans le bloc trajet) lorsqu'on lui demande d'avancer. La longueur d'une case a été fixée à 30 cm (ce qui correspond à un carreau ou un demi carreau du carrelage devant l'amphi). Pour que le robot avance, deux de ses moteurs (moteurs 2 et 3) sont actionnés. Ils s'arrêtent lorsque 39700 créneaux sont passés sur le signal renvoyé par le moteur 3. Ce nombre a été trouvé expérimentalement, en testant où le robot s'arrêtait pour différentes valeurs. Les fonctions qui font tourner le robot sont construites sur le même principe. Les trois moteurs sont actionnés, et s'arrêtent lorsque 23150 créneaux sont passés sur le signal renvoyé par le moteur 3. Cela correspond expérimentalement à une rotation de 90°. Le robot peut tourner dans le sens trigonométrique ou horaire.

3.3 Executer des consignes reçues

Le robot reçoit les consignes sous la forme d'une liste de lettres comme décrite à la partie "trajet". Il lit les instructions et les exécute une par une. La variable `RG_isReady` vaut 1 lorsqu'une instruction est prête à être exécutée. On la fixe à 0 pendant que l'instruction est exécutée.

Le protocole pour exécuter une liste d'instructions calculée à partir du bloc quadrillage est le suivant :

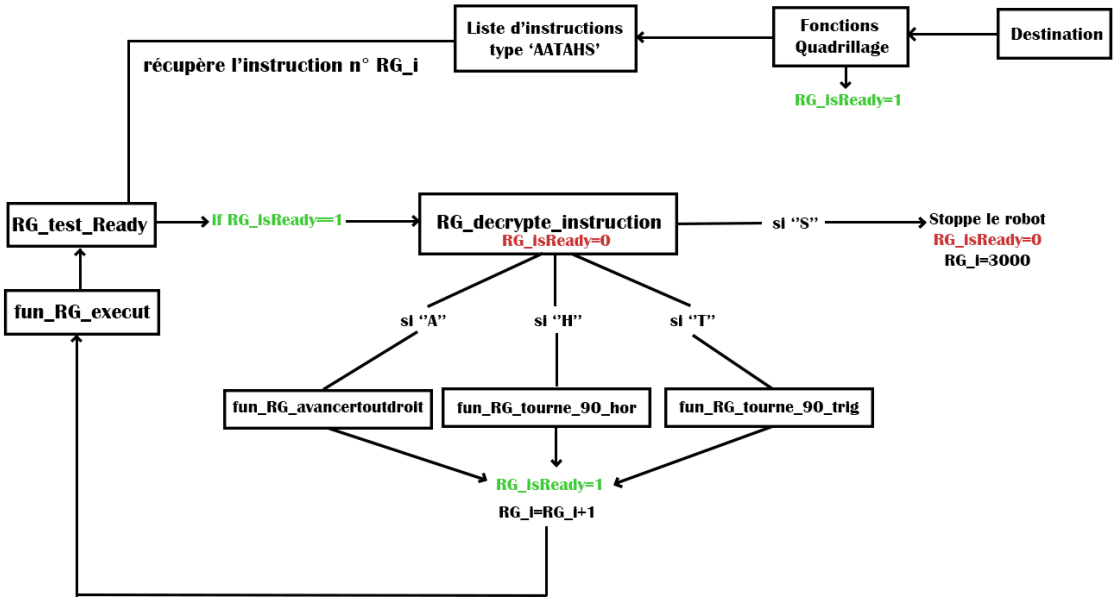

FIGURE 3.2 – Protocole d'éxecution d'une consigne calculée en interne

4 Bloc interface & communication

4.1 Interface

L'interface est faite avec la bibliothèque `tkinter` pour python. L'inclusion d'images est faite avec `PIL`.

Les programmes `colin_interface_s1.py` sera lus avec Python pour afficher l'interface graphique. Il fait appel aux fonctions de `colin_fonctions_interface.py`.

4.1.1 Communication

L'ordinateur supportant le programme python doit être relié à une carte nucléo portant le module bluetooh. Celui-ci étant très proche de celui proposé par le LEnsE et par souci de concision, nous n'indiquerons ici que la fonction `fun_c_transfer_data` qui recopie les données lues sur la liaison série avec l'ordinateur vers la liaison Bluetooth.

4.1.2 Transferts de données

Les transferts de données se font par bloc de deux caractères, le plus souvent une lettre et un chiffre. La lettre correspond le plus souvent au type d'instruction ou à la catégorie de destination souhaitée, et le chiffre à une précision sur cette instruction ou à une destination précise.

5 Fusion des blocs

Cette dernière étape à été amorcée mais non terminée. En effet, la fusion des blocs 'Trajet' et 'Mécatronique' à été effectuée avec succès, mais l'addition du bloc 'Interface&Communication' n'a pas aboutie : les fonctions utilisées pour le Bluetooth et la communication avec python nécessitent en effet des bibliothèques et versions de `MBED` différentes, posant un problème de compatibilité.

Il serait nécessaire

— soit de rechercher des astuces pour outrepasser ce problème de compatibilité (compte tenu du fait que nous n'utilisons jamais deux versions différentes d'une même fonction, mais simplement deux fonction différentes appartenant chacune à une version différente d'une même bibliothèque)

— soit de ré-écrire une partie du code (préférablement du bloc 'Mécatronique') avec une autre bibliothèque de fonction ou la version compatible de la bibliothèque actuelle.

6 Bilan du projet

6.1 Pistes d'amélioration

Certains points du cahier des charges de SOLEC ne sont pas respectés dans le prototype en l'état. Bien que le code en lui-même est fonctionnel, il peut sans doute encore gagner en efficacité. Néanmoins les bases sont posées pour quelques nouvelles fonctionnalités qu'il serait pertinent d'implémenter.

6.1.1 Fiabilité en angle - équilibrage

Le choix empirique du nombre d'incréments d'une roue à compter avant d'arrêter les moteurs dans les fonctions faisant tourner le robot de 90° n'est pas en l'état assez satisfaisant pour satisfaire le cahier des charges. De plus les erreurs angulaires acquises lors d'une rotation dans le sens horaire trigonométriques sont **différentes et de signes opposées**. On peut alors penser que le problème peut venir d'un déséquilibre du robot, potentiellement dû à la batterie utilisée.

En l'état, le programme ne contient qu'une seule fonction permettant d'arrêter le robot lors de sa rotation de 90°, à savoir la fonction `stop_90_mot3` qui est utilisée à la fois pour une rotation dans le sens horaire et dans le sens trigonométrique. Le déséquilibre du robot fait qu'il faut en fait séparer cette fonction en deux, chacune ayant son propre seuil pour marquer la fin de la rotation.

Le protocole pour réduire cette erreur est d'exécuter un nombre important de fois la fonction qu'on souhaite améliorer (`fun_RG_tourne90_trig` ou `fun_RG_tourne90_hor`) de sorte à ce que le robot fasse plusieurs tours complets de 360°. On trace une droite repère pointant vers le centre du robot avant l'exécution des fonctions et une autre après les rotations et on mesure avec un rapporteur l'angle entre les deux traits. On divise ensuite cette valeur par le nombre de tours complets qu'a fait le robot. Si cette erreur est positive, c'est que le robot tourne trop et il faut baisser le seuil dans la fonction correspondante au sens de rotation.

6.1.2 Collection d'information sur le système

Position La transmission `'Q0'` correspond à demander au robot sa position.

La réponse du robot, en forme d'un ou deux blocs de deux caractères, pourrait être `'`xy`'` ou `'X`x`'` ; `'Y`y`'` pour plus de robustesse (avec x et y ses positions `X` et `Y` sur la matrice).

À terme l'interface pourrait même être capable de placer un point rouge sur la carte de l'entrepôt pour indiquer cette position.

Direction Il faudrait également ajouter la question de l'orientation du robot, soit faisant partie de l'information `'Q0'`, soit comme une question à part entière.

La réponse du robot, en forme d'un ou deux blocs de deux caractères, pourrait être `'D`d`'`, avec d un nombre entre 1 et 4 correspondant à la direction.

L'interface devrait être capable de convertir d en une direction compréhensible par l'opérateur, et, à terme, pourrait même placer une flèche rouge sur la carte de l'entrepôt pour indiquer cette orientation.

Niveau batterie La question de la mesure du niveau de batterie est explicite dans le cahier des charges. En ajoutant un ampèremètre, il est possible d'intégrer le courant débité par la batterie et d'en déduire l'autonomie restante. Une valeur standard pour le type de matériel utilisé ici est

La question se pose de l'interprétation de la demande du cahier des charges '1km d'autonomie' étant donné que le robot est amené à tourner et pas seulement avancer. Nous considérons que lors d'un trajet, le ratio du nombre de rotations de 90° et du nombre de déplacement en ligne droite est d'un tiers. Par ailleurs, le robot avance par cases de 30cm ce qui correspond à 32,1 cm de déplacement de chacune des deux roues. Lors d'un tour, chaque roue parcours 18,7cm. Il en résulte que lorsque le robot avance de 60cm dans l'entrepôt (et effectue donc un tour), les roues parcourent en cumulé 184,5cm. Il suffit donc de savoir combien de tours de moteur sont consommés pour parcourir cette distance, et connaissant la consommation du moteur on peut en déduire l'autonomie de la batterie sur 1km.

Un bouton sur l'interface pourraient alors demander le niveau de charge actuel. Dans l'idéal, le robot pourrait aller se charger de lui même lorsque sa batterie devient trop faible avant de reprendre son itinéraire.

6.1.3 Calcul d'un parcours passant par plusieurs cases

Actuellement, le robot se déplace de sa position actuelle à un point B demandé par l'utilisateur.

Un parcours de plusieurs cases à la suite requiert de l'utilisateur qu'il face plusieurs demandes indépendantes - une par case.

Il serait utile de faire évoluer à la fois le bloc 'Interface&Communication' et le bloc 'Trajet' pour que l'utilisateur puisse rentrer une série de cases, demander au robot de s'arrêter pour un temps déterminé sur chaque - ou d'y rester jusqu'à réactivation par l'interface - et que le calcul soit fait automatiquement. Cela demanderait simplement de ré-arranger les fonctions déjà existantes mais sans avoir besoin d'en créer de nouvelles

6.2 Retour d'expérience

Bien que le prototype ne soit pas fini dans sa totalité, l'équipe est très satisfaite du travail qui a été fait pour ce projet. Nous avons pu apprendre, à la fois individuellement en développant nos compétences en programmation ou en robotique, mais aussi collectivement en travaillant ensemble de façon efficace. Notre objectif en début de projet a été de mettre en place un cadre clair de travail en équipe permettant à chaque membre d'avoir accès à tout ce qui est réalisé mais aussi permettant de limiter au maximum les problèmes lors de la mise en commun des différents blocs. Sur ce dernier point, nous pouvons nous satisfaire que la mise en commun des blocs "mécatronique" et "trajets" a été faite sans problème majeur. La mise en commun avec le bloc "interface & communication" est plus délicate du fait de la différence de version du langage de programmation utilisé, mais nous ne pensons pas qu'il s'agisse d'un obstacle majeur.

Ainsi, bien que le prototype ne soit pas complet, nous considérons ce projet réussi car nous pensons qu'il est suffisamment structuré et expliqué pour pouvoir être repris par un autre groupe et mené à bout sans incompréhension majeur du travail que nous avons réalisé, comme en atteste ce document.

Annexes

Programmes informatiques

Tous les programmes présentés sont basés sur des travaux de Julien Villemejane accessibles via le site du LEnsE

Interface humain-machine

`colin_interface_s1.py`

Ce programme est à lancer dans python pour démarrer l'interface graphique. La console de python sera utilisée pour lire les retours du système.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 25 16:24:24 2023

@author: colin.guirardel
"""

# coding: utf-8

"""
"""
bat = 1 # variable représentant la batterie restante

tab_destinations=["P"+str(i+1) for i in range(7)]+["A"+str(i+1) for i
    in range(8)]+["F"+str(i+1) for i in range(6)]+["C"+str(i+1) for i
    in range(2)]+["L"+str(i+1) for i in range(2)]+["R0","F0"]


"""
fonctions
"""
def fun_c_retour_base() :
    fun_c_send_data("R0",flag_port,serNuc)

def fun_c_demande_pos() :
    fun_c_send_data("Q0",flag_port,serNuc)
    fun_c_lect_tampon1()

def fun_c_envoi_destination():
    fun_c_send_data(menu_destinations.get(),flag_port,serNuc)


def fun_c_choix_port():
        # To select the port to use
        selectPort =  menu_ports.get() #de la forme "COM1: Port de
            communication (COM1)"
        index=str(selectPort).find(":")
        numPort=selectPort[3:index]
        print(numPort)
        print(f"Port Selected : COM{numPort}")
        # To open the serial communication at a specific baudrate
        global serNuc
        if not serNuc is None:
            serNuc.close()
        serNuc = Serial('COM'+str(numPort), 9600)  # Under Windows only
        print(serNuc)
        global flag_port
        flag_port="up"
```

```
def fun_c_lect_tampon1():
    fun_c_lect_tampon(flag_port, serNuc)


def fun_c_read_serial():

        print(serNuc.read(2))
"""
"""

import tkinter
from tkinter import *
import tkinter.ttk as ttk
from tkinter.filedialog import *
from tkinter.messagebox import *
import PIL
from PIL import ImageTk, Image

from serial import Serial
import serial.tools.list_ports

from colin_fonctions_interface import *


fenetre = Tk()


fenetre.geometry("400x550")

global flag_port
flag_port = "down"


global serNuc
serNuc= None
# Create a photoimage object of the image in the path
image1 = PIL.Image.open("C:/Users/colin.guirardel/PROTIS/cropped-
    soleclogo-1.png")
img_solec = ImageTk.PhotoImage(image1)

image2 = PIL.Image.open("C:/Users/colin.guirardel/PROTIS/plan.png")
img_map = ImageTk.PhotoImage(image2)



label_sol = Label(fenetre, image=img_solec)
label_sol.pack()

#panneau ports
p_ports = PanedWindow(fenetre, orient=VERTICAL)
p_ports.pack( expand=Y, fill=BOTH, pady=10, padx=10)
#p_ports.add(Button(p_ports, text="Lecture", command=fun_c_read_serial,
     anchor=CENTER))

ports = serial.tools.list_ports.comports()



# To obtain the list of the communication ports
tab_ports=[]
```

```
for port, desc, hwid in sorted(ports):
    tab_ports.append("{}: {}".format(port, desc))
menu_ports = ttk.Combobox(p_ports, values=tab_ports)
menu_ports.current(0)
p_ports.add(menu_ports,padx=50)
p_ports.add(Button(p_ports, text="Choix du port", command=
    fun_c_choix_port, anchor=CENTER),padx=120)
p_ports.add(Button(p_ports, text="Lecture tampon", command=
    fun_c_lect_tampon1, anchor=CENTER),pady=10,padx=120)
p_ports.pack()



#panneau consigne
menu_destinations = ttk.Combobox()
p_cons = PanedWindow(fenetre, orient=VERTICAL)
p_cons.pack( expand=Y, fill=BOTH, pady=10, padx=10)
p_cons.add(Label(p_cons, text = "Selectionnez une destination", anchor=
    CENTER))
menu_destinations = ttk.Combobox(p_cons, values=tab_destinations)
menu_destinations.current(0)
p_cons.add(menu_destinations,padx=120)
p_cons.add(Button(p_cons, text="Envoi consigne", command=
    fun_c_envoi_destination, anchor=CENTER),padx=120)
p_cons.add(Label(p_cons, image=img_map, anchor=CENTER))
p_cons.pack()


#panneau batterie
p_bat = PanedWindow(fenetre, orient=VERTICAL)
p_bat.pack( expand=Y, fill=BOTH, pady=10, padx=10)
#p_bat.add(Label(p_bat,text="Batterie "+str(bat*100)+"%", bg="white",
    anchor=CENTER),padx=150)
p_bat.add(Button(p_bat, text="Retour a la base", command=
    fun_c_retour_base, anchor=CENTER, bg="orange"),padx=120)
p_bat.add(Button(p_bat, text="Demander position", command=
    fun_c_demande_pos, anchor=CENTER),padx=120)
p_bat.pack()


fenetre.mainloop()
serNuc.close()
```

`colin_fonctions_interface.py`
Ce programme comprends les fonctions utilisées par `colin_interface_s1.py`

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb  8 16:31:22 2023

@author: colin.guirardel
"""
from tkinter import *
import tkinter.ttk as ttk
from tkinter.filedialog import *
from tkinter.messagebox import *


from serial import Serial
import serial.tools.list_ports

from colin_fonctions_interface import *

def fun_c_liste_ports():
    if __name__ == "__main__":
        ports = serial.tools.list_ports.comports()
        # To obtain the list of the communication ports
        for port, desc, hwid in sorted(ports):
            print("{}: {}".format(port, desc))




def fun_c_send_data(data_to_send, flag_port, serNuc):
    if flag_port == "down" :
        showwarning("Choix du port","Veuillez choisir un port de
            communication")
    else :
        serNuc.write(bytes(data_to_send, 'ascii'))
        data_rec = serNuc.read(2)  # bytes
        print(str(data_rec))


def fun_c_lect_tampon(flag_port, serNuc):
    if flag_port == "down" :
        showwarning("Choix du port","Veuillez choisir un port de
            communication")
    else :
        data_rec = serNuc.read(serNuc.in_waiting)  # bytes
        print(str(data_rec))

def errortest():
    error("test concluant !")
```

`fun_c_transfer_data`
Cette fonction permet le transfert des données venant de l'interface vers robot.

```
// Fonction de transfert du pc vers le module BT nRF24L01
void fun_c_transfer_data(void){
    /* Lecture donnée depuis nRF24 */
    if ( nRF24_mod.readable() ) {

        // ...read the data into the receive buffer
        rxDataCnt = nRF24_mod.read( NRF24L01P_PIPE_P0, dataReceived,
            TRANSFER_SIZE);

        // Display the receive buffer contents via the host serial link
        //debug_pc.printf("\tD = ");
        for ( int i = 0; i < rxDataCnt; i++ ) {
            debug_pc.printf("%x", dataReceived[i]);
        }
        //debug_pc.printf("\r\n");
    }
    /* Transmission donnée depuis nRF24 */
    while ( not debug_pc.readable() ) {
        }
        data1=debug_pc.getc();
    while ( not debug_pc.readable() ) {
        }
        data2=debug_pc.getc();
        nRF24_mod.setRfFrequency(2400);
        tab_c_data[0]=data1;
        tab_c_data[1]=data2;
        nRF24_mod.write( NRF24L01P_PIPE_P0, tab_c_data,TRANSFER_SIZE );
        debug_pc.printf("%c%c", data1, data2);
        //debug_pc.printf( "OK");
        wait_us(100000);
}
```