Rapport technique

BASSO-LACROIX Stella **HIRSCHAUER Noé** **LIVET Raphaël** **MOULIGNEAUX Nathan**

1 - Introduction

1.1 - Rappel objectifs + schéma principe

Le but du projet est de concevoir un robot censé aller sur Mars. Il doit pouvoir se déplacer et recevoir les commandes de déplacement par Bluetooth. Il doit également être équipé de capteurs de thermomètre et de température. Les mesures doivent être effectuées tous les 10 cm et envoyées à la base toutes les 10 minutes.

L'interface Humain Machine permettant de transmettre les ordres de parcours doit être utilisable sans formation préalable et les données doivent être affichées sous forme de graphe en fonction de la distance ou du temps.

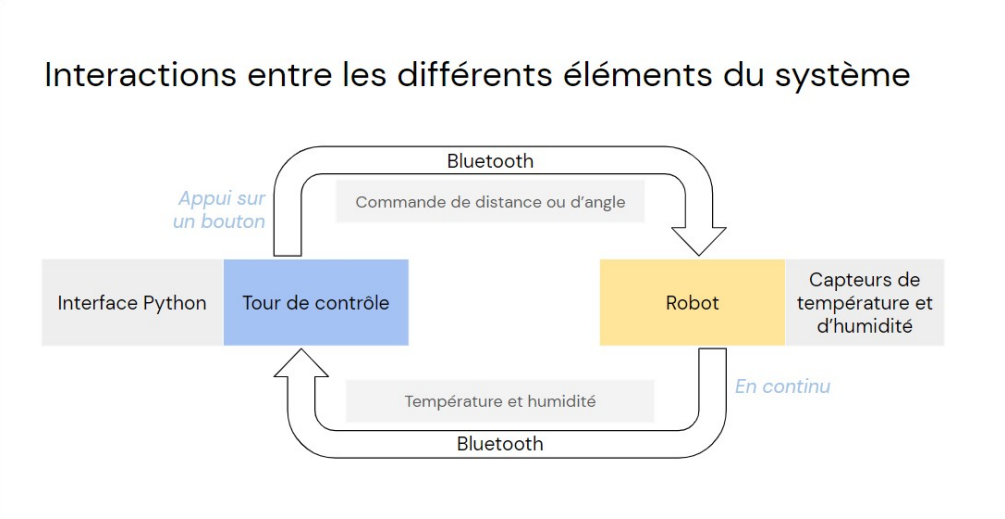


Schéma de principe du robot

1.2 - Cahier des charges / Contraintes et performances attendues

Rapidité : La vitesse de déplacement doit être comprise entre 10 et 30cm/s.

Fiabilité : Erreur maximale de : 2cm sur la position, 3° sur l'angle

Autonomie : Le robot doit pouvoir parcourir 1km avant d'être rechargé.

Ergonomie : L'interface doit pouvoir être utilisé par quelqu'un sans formation préalable.

1.3 - Notice d'utilisation

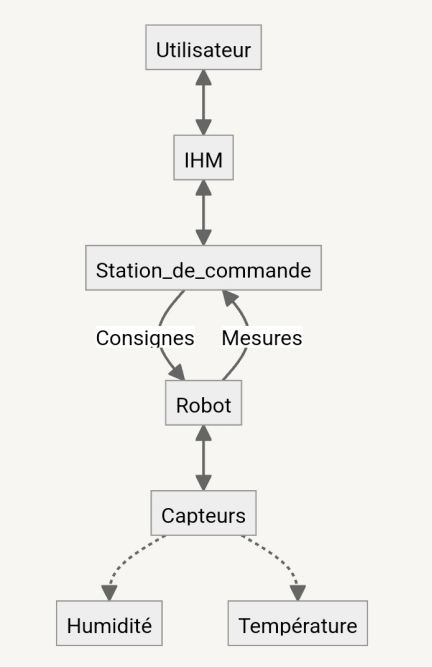
Le centre de contrôle est constitué d'un ordinateur portable muni d'une interface Python et d'une carte Nucléo.

Pour envoyer une consigne de distance ou de rotation, il faut rentrer la valeur sur l'interface Python, cliquer sur le bouton 'Envoi' correspondant de l'interface et maintenir enfoncer le bouton de la Nucléo.

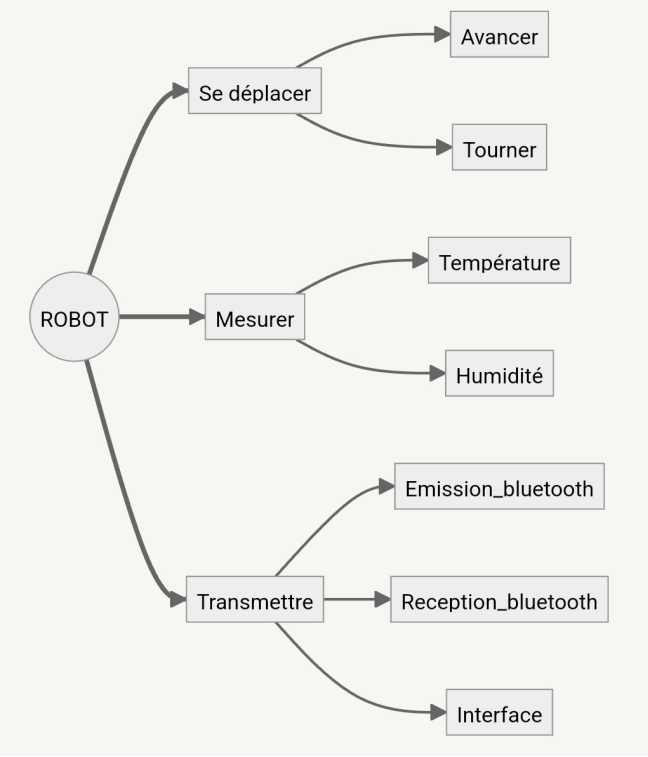
Dans l'état du projet actuel, il n'est pas possible de demander au robot de parcourir deux fois la même distance ou la même rotation de manière consécutive.

Pour la réception des mesures, il faut appuyer sur le bouton de la carte puis appuyer sur le bouton "Mesures" de l'interface.

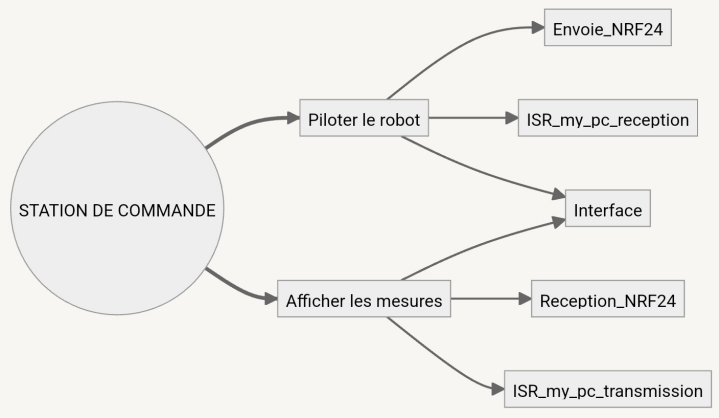
1.4 - Diagramme des éléments



1.5 - Schéma fonctionnel du robot



1.6 - Schéma fonctionnel de la station de commande



2 - Description

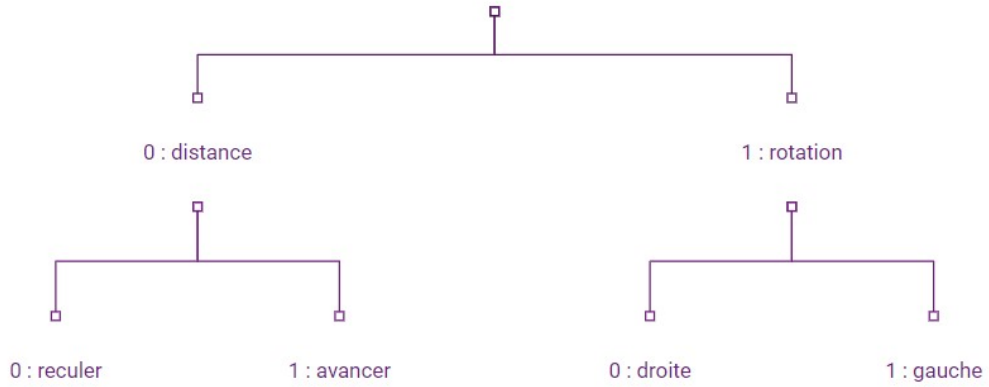
2.1 - Interface Python

2.2.1 - Fonctionnement général

L'interface Python doit permettre de choisir les commandes de déplacement et les envoyer à la carte Nucléo du robot.

L'application envoie systématiquement 3 valeurs à la carte Nucléo.

Les deux premières sont 0 ou 1 en fonction de si l'on effectue une rotation ou une translation et de leur sens, la dernière correspond à la valeur entrée dans l'interface.



Choix des valeurs indiquant quel déplacement est effectué

Le bouton mesure permet l'affichage des données envoyé par la carte Nucléo à l'interface Python. Elles s'affichent dans le shell.

A cause du traitement des données par la carte Nucléo, on ne peut envoyer que des consignes de distance et d'angle comprises entre 10 et 99.

Le programme envoie correctement les valeurs entrées.

2.1.2 - Code

```
# importing libraries
# Librairie interface:
from PyQt5.QtWidgets import *
from PyQt5 import QtCore, QtGui
from PyQt5.QtGui import *
from PyQt5.QtCore import *
import sys

#Librairie lien nucléo :
from serial import Serial
import serial.tools.list_ports

from PyQt5.uic import loadUi

class Window(QMainWindow):

    def __init__(self):
        super().__init__()

        # setting title
        self.setWindowTitle("Robot Véronica")

        # setting geometry
        self.setGeometry(100, 100, 600, 400)

        # calling method
        self.UiComponents()

        # showing all the widgets
        self.show()

        # To obtain the list of the communication ports
        self.ports = serial.tools.list_ports.comports()
        self.ports.sort()

        # To clear the list on the window
        #self.ports.clear()
        # To add all the communication ports to the list
        for port, desc, hwid in sorted(self.ports):
            print(f"{port} [{desc}]")

        self.selectPort = 16 #input('COM ?'), A REMPLIR AVANT DE LANCER LE PROGRAMME

        # To create an empty serial connection
        self.serNuc = Serial()
        self.connected = 0
        self.connectToNucleo()
        self.value = 0
        self.value2 = 0
        self.mesure_humidite = 0
        self.mesure_temp = 0

        #Connexion du bouton
        self.button_distance.clicked.connect(self.Envoiedonnees_distance)
        self.button_angle.clicked.connect(self.Envoiedonnees_angle)
        self.button_recup_mesure.clicked.connect(self.affiche_mesure)

    # method for widgets
    def UiComponents(self):
        # Curseur pour la distance
        self.spin = QSpinBox(self)
        self.spin.setRange(-99, 99)
        self.spin.setSingleStep(1)
        self.spin.setSuffix(' cm')

        # setting geometry to spin box
        self.spin.setGeometry(100, 100, 100, 40)

        # adding action to the spin box
        self.spin.valueChanged.connect(self.show_result)

        # creating label show result
        self.label = QLabel(self)
```

```
        # setting geometry
        self.label.setGeometry(100, 200, 200, 40)

        # Curseur pour l'angle

        # creating spin box2
        self.spin2 = QSpinBox(self)

        self.spin2.setRange(-99, 99)
        self.spin2.setSingleStep(1)
        self.spin2.setSuffix(' °')

        # setting geometry to spin box
        self.spin2.setGeometry(300, 100, 100, 40)

        # adding action to the spin box
        self.spin2.valueChanged.connect(self.show_result2)

        # creating label show result
        self.label2 = QLabel(self)

        # setting geometry
        self.label2.setGeometry(300, 200, 200, 40)

        # creating label show result
        self.label3 = QLabel(self)

        # setting geometry
        self.label3.setGeometry(250, 300, 200, 40)


        self.button_distance = QPushButton('Envoi', self)
        self.setToolTip('This is an example button')
        self.button_distance.move(100,70)

        self.button_angle = QPushButton('Envoi', self)
        self.setToolTip('This is an example button')
        self.button_angle.move(300,70)

        self.button_recup_mesure = QPushButton('Mesure', self)
        self.setToolTip('This is an example button')
        self.button_recup_mesure.move(200,200)

    def __del__(self):
        self.serNuc.close()

    def connectToNucleo(self):
        if(self.connected == 0):
            self.serNuc = Serial("COM"+str(self.selectPort), 115200)
            self.connected = 1

    def quitApp(self):
        self.serNuc.close()
        self.close()

    def Envoiedonnees_distance(self): #Fonction d'envoi des données de distance
        print(self.connected)
        if(self.connected):
            print(str(self.value))
            self.serNuc.write(bytes('0', 'utf-8'))
            if (self.value>0):
                self.serNuc.write(bytes('1', 'utf-8')) #avancer
            else:
                self.serNuc.write(bytes('0','utf-8')) #reculer
            self.serNuc.write(bytes(str(abs(self.value)),'utf-8'))

    def Envoiedonnees_angle(self): #Fonction d'envoi des données de rotation
        print(self.connected)
        if(self.connected):
            print(str(self.value2))
            self.serNuc.write(bytes('1','utf-8'))
            if (self.value>0):
                self.serNuc.write(bytes('1', 'utf-8'))
            else:
                self.serNuc.write(bytes('0','utf-8'))
            self.serNuc.write(bytes(str(abs(self.value2)),'utf-8'))
```

```
    #fonction qui récupère la distance à parcourir
    def show_result(self):
        # getting current value
        self.value = self.spin.value()
        # setting value of spin box to the label
        self.label.setText("Distance : " + str(self.value))

    # fonction qui récupère l'angle
    def show_result2(self):
        # getting current value
        self.value2 = self.spin2.value()
        # setting value of spin box to the label
        self.label2.setText("Angle : " + str(self.value2))

    #fonction qui récupère les mesures :
    def affiche_mesure(self):
        while(self.serNuc.in_waiting == 0):
            print("Wait !")
        rec_data_nucleo = self.serNuc.read(2)
        self.mesure_humidite = rec_data_nucleo[0]
        self.mesure_temp = rec_data_nucleo[1]
        print(f'Température = {rec_data_nucleo[0]}')
        print(f'Humidité = {rec_data_nucleo[1]}')


# create pyqt5 app
App = QApplication(sys.argv)
# create the instance of our Window
window = Window()
window.show()

# start the app
sys.exit(App.exec())
```

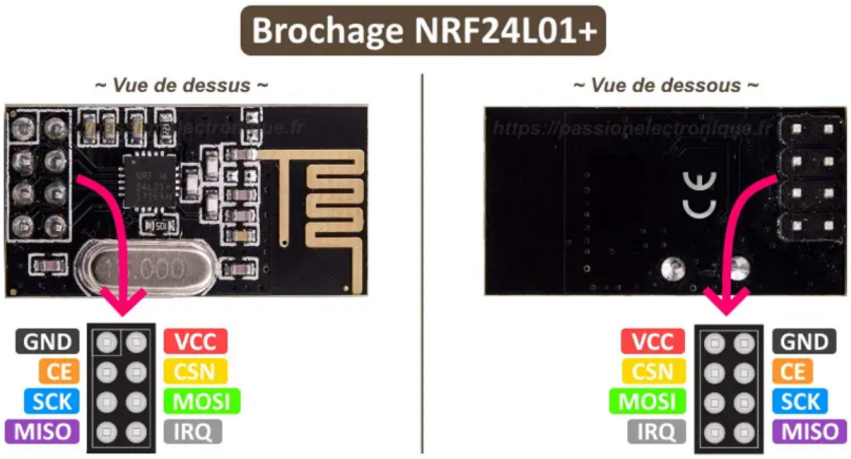
2.2 - Traitement des données envoyées par l'interface Python par la Nucléo

2.2.1 - Fonctionnement général

Lorsque le bouton est enfoncé, la carte Nucléo lit les données envoyées par l'interface Python. La carte lit caractère par caractère, on effectue donc 4 `getc` pour récupérer toutes les données. On concatène ensuite les deux dernières afin de reformer la commande initiale. Ces actions sont effectués par la fonction `ISR_my_pc_reception`.

Ces données sont ensuite mises dans un tableau qui est envoyé en Bluetooth à la carte du robot par l'intermédiaire de la fonction `Envoie_NRF24`.

Pour envoyer en bluetooth, on utilise le module bluetooth NRF24L01+ dont le schéma de branchement est le suivant :



Cependant, un emplacement prévu au bon fonctionnement du module est préexistent sur la carte triangulaire verte utilisé pour controler les moteurs. On remarque tout de même que sans cette emplacement, les branchements par cable sur le module NRF24L01+ entrainaient de nombreuses erreurs dans l'émission et la réception du bluetooth, surement à cause de faux contacts. On en déduit que le module est très sensible à son cablage.

Pour la réception de mesures, la carte lit ce qui est envoyé en bluetooth par l'intermédiaire de la fonction `Reception_NRF24` détaillé plus tard. Lorsque le bouton de la carte est appuyé, les mesures lues via la liaison Bluetooth sont lues, traitées et envoyer à l'interface Python via la fonction `ISR_my_pc_transmission`.

2.2.2 - Codes

```
#include "mbed.h"
#include "MOD24_NRF.h"
#include "nRF24.h"

Serial          debug_pc(USBTX, USBRX);
InterruptIn     bp_int(USER_BUTTON);
void ISR_my_pc_reception(void);
void ISR_my_pc_transmission(void);

/* Main Function */
int main() {
    debug_pc.printf("Test\r\n");
    initNRF24();

    while(1) {
        Envoie_NRF24();
        debug_pc.baud(115200);
        debug_pc.attach(&ISR_my_pc_reception);
        Reception_NRF24();
        bp_int.fall(&ISR_my_pc_transmission);
    }
}

void ISR_my_pc_reception(void){
    if(debug_pc.readable()){
         data1=debug_pc.getc();
         data2=debug_pc.getc();
         data3 = debug_pc.getc();
         data4=debug_pc.getc();
         data = (data3-48)*10 + data4-48  ;

            dataToSend[0]= data1-48;
            dataToSend[1]= data2-48;
            dataToSend[2]= data; }
}

void ISR_my_pc_transmission(void){
```

```
    if (debug_pc.writable()){
        debug_pc.putc(dataReceived[0]*20/255+10 );   //Affichage de la température
        debug_pc.putc( dataReceived[1]*20.0/255+30); //Affichage de l'humidité*20/255+30
    }
}
```

Bibliothèque “bluetooth”

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/268b9015-f8b7-4998-8132-d72d8dedfe7f/bluetooth.cpp

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/e4ef646e-e3b7-4007-aa20-df05ea3d0d91/bluetooth.h

On note que la réception des mesures n’est pas opérationnelle à la fin du projet.

2.3 - Exécuter une commande déplacement rectiligne

2.3.1 - Description de la fonction

Le but de cette fonction est de faire effectuer au robot un déplacement rectiligne sur une distance donné comme variable de code (qui sera dans l’intégration finale fournie en bluetooth par l’ordinateur de contrôle).

Pour ce faire, nous utilisons les deux moteurs pilotés par un code MBED via la carte NUCLEO, à l’aide du codeur incrémental dont dispose chacun d’eux. Nous déduisons des constantes de notre système $($ rayons des roues R, et nombre d’incréments du compteur par tour $N)$ la commande (en nombre d’incréments a effectuer sur les roues) la distance D parcourue pour un incrément du codeur :

$$D = 2\pi R/N$$

Nous pouvons alors traduire la distance à parcourir par le robot par un nombre d’incréments au bout duquel il doit s’arrêter.

Notons que nos deux moteurs, bien qu’étant de même modèle, ne fonctionnent pas exactement de la même manière (l’un est paefois plus rapide que l’autre par exemple). Aussi avons-nous été amenés à mettre en place un asservissement sur la vitesse de chacun deux afin de garantir que les deux roues aillent aussi vite l’une que l’autre, et donc que notre robot se déplace bien en ligne droite.

Nous fournissons et expliquons ci-dessous le code implémenté pour effectuer ce déplacement rectiligne. Pour faciliter l’intégration dans le code final, ce code, ainsi que celui utilisé pour la rotation, on été mis sous la forme de deux fonctions regroupées dans un bibliotèque nommée `deplacement.h`.

2.3.2 - Utilisation

Pour faire avancer le robot en ligne droite il suffit de faire appel à la fonction suivante :

```
Avancer(int Distance,double vitesse);
```

`Avancer()` prend deux arguments :

1. `Distance` de type `int` est la distance à parcourir en cm
2. `vitesse` de type `double` est le rapport cyclique des moteurs, doit être compris entre 0 et 1

2.3.3 - Explication du code

Variables globales

Pour fonctionner correctement des variables globales sont définies pour que les différentes fonctions puissent récupérer les valeurs qui sont affectées lors de l'initialisation. Elles sont détaillées ci dessous.

```
#define N   115             // Nombre d'impulsions pour un tour de roue
#define R   6.25            // Rayon de la roue
#define d   25.5            // Distance entre les deux roues


// Bornes reliées aux moteurs des roues (les roues sont à l'AVANT)
PwmOut moteur_droite_R(PB_14);      //moteur droit sens reculer
PwmOut moteur_droite_A(PB_13);      //moteur droit sens avancer
PwmOut moteur_gauche_A(PB_8);       //moteur gauche sens avancer
PwmOut moteur_gauche_R(PC_9);       //moteur gauche sens reculer

// Fonctions d'interruption permettant de détecter la rotation des roues
InterruptIn codeur_gauche(PB_4);
InterruptIn codeur_droite(PB_2);

// Variables d'asservissement en direction
double erreur = 0;          //ecart angulaire entre les deux roues
double Ierreur = 0;         //Integrale de l'erreur
double Kp = 0.05;           //Gain correcteur proportionnel
double Ki = 0.00001;        //Gain correcteur intégral
double rc;                  // Rapport cyclique

// Variables de consigne (distance ou rotation)
double  distance_c;         //consigne de distance en cm
double  distance_p = 0;     //distance parcourue
int     sens;               //1 si le robot va vers l'avant
double  theta_c;            //consigne de rotation en degrés

// Variables de position angulaire de chaque roue
double angle_gauche=0;      //roue gauche
double angle_droite=0;      //roue droite
double fcorr;               //facteur de correction pour l'inertie

// Résolution
double delta_d = (2*3.1415926535*R)/(N);   //distance parcourue pour 1 impulsion
double delta = (360*R)/(d*N);              // Angle réel du robot pour 1 impulsion
```

Initialisation

La fonction `Avancer()` initialise les moteurs et déclare des fonctions d'interruptions sur les capteurs angulaires des moteurs. Son code est donné ci dessous :

```
void Avancer(int distance_commande, double vit){

    rc = vit;                           //affectation du rapport cyclique
    distance_c=distance_commande;       //affectation de la distance de consigne

    //Initialisation des moteurs
    moteur_droite_R.period_ms(10);
    moteur_droite_A.period_ms(10);
    moteur_gauche_A.period_ms(10);
    moteur_gauche_R.period_ms(10);

    moteur_droite_R.write(0);           //le robot est à l'arrêt
```

```
    moteur_droite_A.write(0);
    moteur_gauche_A.write(0);
    moteur_gauche_R.write(0);

    //Interruptions
    codeur_gauche.rise(&avancer_gauche);
    codeur_gauche.fall(&avancer_gauche);
    codeur_droite.rise(&avancer_droite);
    codeur_droite.fall(&avancer_droite);

    if (distance_c>0){sens=1;}          //affectation du sens
    else{sens=0;}

    // Permet de démarrer le mouvement
    Demarrer();
}
```

La commande en vitesse des moteurs se fait par affectation d'un rapport cyclique. Pour déterminer le sens de rotation, il faut choisir sur quelle borne on envoie le signal. Par exemple, si on affecte `rc` a `moteur_droite_A` et `0` à `moteur_droite_R` alors le moteur droit tournera dans le sens qui correspond à l'avant du robot avec un rapport cyclique de `rc`. Il faut échanger les valeurs si l'on veut faire tourner le moteur dans l'autre sens.

Les fonctions d'interruptions sont déclenchées sur les fronts montants et descendants du signal d'un des codeurs rotation de chaque roue pour faire une actualisation plus rapide et mieux asservir en direction.

Démarrage

La fonction `Demarrer()` permet juste de lancer le mouvement du robot dans le bon sens par rapport à la distance de consigne. Une fois que les moteurs ont commencé à tourner, ce sont les fonctions d'interruption qui prennent le relais pour contrôler le déplacement.

```
void Demarrer(void){

    //Renitialisation
    angle_gauche=0;
    angle_droite=0;
    distance_p=0;

    //Lance le mouvement
    if (sens==1){
        moteur_droite_R.write(0);
        moteur_droite_A.write(rc);
        moteur_gauche_A.write(rc);
        moteur_gauche_R.write(0);
    }
    else{
        moteur_droite_R.write(rc);
        moteur_droite_A.write(0);
        moteur_gauche_A.write(0);
        moteur_gauche_R.write(rc);
    }
```

Asservissement

L'objectif de l'asservissement est de faire aller le robot en ligne droite et de l'arrêter une fois que la distance parcourue correspond à la distance de consigne. Pour cela on utilise deux fonctions d'interruptions : `avancer_droite()` et `avancer_gauche()`.

```
void avancer_droite(void){
    angle_droite = angle_droite + 1;    //Nombre d'impulsions de la roue droite
}
```

Cette fonction ne fait que compter le nombre d'impulsions délivrées par le capteur angulaire de la roue droite. Cette valeur est utilisée dans `avancer_gauche()` pour s'assurer que le nombre d'impulsions à droite est bien égal au nombre d'impulsions à gauche, ce qui veut dire que les roues ont parcouru la même distance et que le robot est allé en ligne droite :

```
void avancer_gauche(void){
  angle_gauche = angle_gauche + 1;          //Nombre d'impulsions de la roue gauche
    distance_p = distance_p + delta_d;      //Distance parcourue par le robot

    //Condition d'arret des moteurs : on a parcouru la bonne distance
    if(1.0*abs(distance_p)>=0.9*abs(distance_c)){

        //Envoi d'une impulsion pour freiner puis arret
        if (sens==1){
            moteur_gauche_A.write(0);
            moteur_gauche_R.write(1);
            moteur_droite_R.write(1);
            moteur_droite_A.write(0);
        }
        if (sens==0){
            moteur_gauche_A.write(1);
            moteur_gauche_R.write(0);
            moteur_droite_R.write(0);
            moteur_droite_A.write(1);
        }

        moteur_gauche_A.write(0);
        moteur_gauche_R.write(0);
        moteur_droite_R.write(0);
        moteur_droite_A.write(0);

    }

    else {
        erreur = angle_droite-angle_gauche;     //ecart angulaire entre les roues
        Ierreur += erreur;                      //intégrale de l'erreur

            // On adapte le rapport cyclique des roues (proportionnel intégral)
            if (sens==1){
                moteur_droite_A.write(rc - Kp*erreur - Ki*Ierreur);
                moteur_gauche_A.write(rc + Kp*erreur + Ki*Ierreur);
                moteur_droite_R.write(0);
                moteur_gauche_R.write(0);
            }
            if (sens==0){
                moteur_droite_R.write(rc - Kp*erreur - Ki*Ierreur);
                moteur_gauche_R.write(rc + Kp*erreur + Ki*Ierreur);
                moteur_droite_A.write(0);
                moteur_gauche_A.write(0);
            }
    }
}
```

Cette fonction compte le nombre d'impulsions de la roue gauche et incrémente la distance parcourue au fur et à mesure. La suite peut se séparer en 2 parties :

1. Si l'on a parcouru la bonne distance, on envoie une impulsion dans le sens contraire au déplacement pour freiner le robot puis on éteint les moteurs. Il y a un facteur 0.9 pour prendre en compte l'inertie du robot qui continu à rouler après que le moteurs aient été coupés
2. Sinon, on compare la position angulaire des deux roues et on adapte le rapport cyclique en fonction. Si la roue droite est en avance sur la roue gauche, la variable `erreur` est positive et on diminue le rapport cyclique de la roue droite tout en augmentant le rapport cyclique de la roue gauche pour compenser l'écart. La correction intégrale permet de lisser les effets de la correction proportionnelle pour éviter que le robot avance en zig-zag.

2.3.4 - Test de validation

La méthode de test utilisée est la suivante : placer le robot à côté d'un mètre et mesurer la distance parcourue afin de vérifier si elle correspond bien à la commande fournie. Bien que simple, elle permet d'évaluer la précision de notre déplacement au centimère, ce qui est assez pour vérifier si le critère de précision du cahier des charges est respecté ou non.

2.4 - Exécuter une commande de rotation

2.4.1 - Description de la fonction

Le but de cette fonction est de faire effectuer au robot une rotation d'un angle donné comme variable de code (qui sera dans l'intégration finale fourni en bluetooth par l'odrinateur de contrôle).

Pour ce faire, nous utilisons les deux moteurs pilotés par un code MBED via la carte NUCLEO, à l'aide du codeur incrémental dont dispose chacun d'eux. Nous déduisons des constantes de notre système $\left(\text{distance entre les routes } d\text{, rayons de celles-ci } R\text{, et nombre d'incréments du compteur par tour } N\right)$ la commande (en nombre d'incréments a effectuer sur les roues) correspondant à une rotation d'un angle θ_{consigne} :

$$\boxed{N_{\text{consigne}} = \frac{d}{R}\frac{\theta^{\circ}_{\text{consigne}}}{360}N}$$

Le code implémenté à partir de cette formule est décrit et détaillé ci-dessous.

2.4.2 - Utilisation

Pour faire tourner le robot d'un angle en degrés, il suffit de faire appel à la fonction suivante :

```
Tourner(double theta, double vit);
```

`Tourner()` prend deux arguments :

1. `theta` de type `int` est la distance à parcourir en cm
2. `vi` de type `double` est le rapport cyclique des moteurs, doit être compris entre 0 et 1

2.4.3 - Explication du code

Les variables globales utilisées pour la rotation ont déjà été définies dans la partie 2.3.3

Initialisation

La fonction `Tourner()` initialise les moteurs et déclare les fonctions d'interruption. Son code est donné ci dessous.

```
void Tourner(double theta, double vit){

    //Affectation des variables globales
    rc = vit;
    theta_c = theta;

    //Initialisation
    moteur_droite_R.period_ms(10);
    moteur_droite_A.period_ms(10);
    moteur_gauche_A.period_ms(10);
    moteur_gauche_R.period_ms(10);

    angle_gauche=0;
    angle_droite=0;
```

```
    //Interruptions
    codeur_gauche.rise(&tourner_gauche);
    codeur_gauche.fall(&tourner_gauche);
    codeur_droite.rise(&tourner_droite);
    codeur_droite.fall(&tourner_droite);

    //Détermination du sens de rotation
    if(theta_c>0){sens = 0;}
    else{sens = 1;}

    //Déterminatin du facteur de correction d'inertie (empirique)
    if(abs(theta_c)<=90){
        fcorr=0.67;
        }
    if((abs(theta_c)>90)&(abs(theta_c)<180)){
        fcorr=(0.9-0.67)*(abs(theta_c)-90)/90+0.67;
        }
    if(abs(theta_c)>180){
        fcorr=1;
        }

    //Début de la rotation dans le bon sens
    if (sens==1){
        moteur_droite_R.write(0);
        moteur_droite_A.write(rc);
        moteur_gauche_A.write(0);
        moteur_gauche_R.write(rc);
    }
    else{
        moteur_droite_R.write(rc);
        moteur_droite_A.write(0);
        moteur_gauche_A.write(rc);
        moteur_gauche_R.write(0);
    }
}
```

On détermine le facteur de correction d'inertie `fcorr` qui prend en compte le fait que le robot a de l'élan lorsqu'il tourne et quand les moteurs sont coupés la rotation se continue légèrement. Ce facteur a été déterminé de façon empirique.

Les fonctions d'interruptions `tourner_gauche` et `tourner_droite` sont déclarées sur les fronts montants et descendants d'un seul des codeurs de rotation de chaque moteur. Ce sont elles qui contrôleront l'arrêt de la rotation

La rotation est démarrée à la fin de la fonction. On fait tourner une roue dans un sens et l'autre roue dans l'autre sens pour que le robot tourne sur place. Une fois que les moteurs sont en mouvement, les fonctions d'interruptions sont déclenchées et prennent le relais dans le contrôle du robot.

Rotation

Les fonctions d'interruptions sont symétriques sur les deux moteurs. Elles comptent le nombre d'impulsions délivrées par les codeurs de rotation et la convertissent en angle réel du robot avec le facteur `delta` initialisé dans les variables globales.

Moteur gauche

```
void tourner_gauche(void){

    //Angle réel du robot
    angle_gauche=angle_gauche+delta;

    //Condition d'arrêt des moteurs
    if(fcorr*abs(theta_c/2)<=1.0*abs(angle_gauche)){
        // On envoie une impulsion pour arreter les moteurs puis on les éteint
        if (sens==1){
            moteur_gauche_A.write(rc);
            moteur_gauche_R.write(0);
        }
```

```
        if (sens==0){
            moteur_gauche_A.write(0);
            moteur_gauche_R.write(rc);
        }
        moteur_gauche_A.write(0);
        moteur_gauche_R.write(0);
    }
}
```

Moteur droit

```
void tourner_droite(void){
    //Angle réel du robot
    angle_droite=angle_droite+delta;

    //Condition d'arrêt des moteurs
    if(fcorr*abs(theta_c/2)<=1.0*abs(angle_droite)){
       // On envoie une impulsion pour arreter les moteurs puis on les éteint
        if (sens==1){
            moteur_droite_R.write(rc);
            moteur_droite_A.write(0);
        }
        if (sens==0){
            moteur_droite_R.write(0);
            moteur_droite_A.write(rc);
        }
        moteur_droite_R.write(0);
        moteur_droite_A.write(0);
    }
}
```

La condition d'arrêt est que lorsque l'angle réel du robot (corrigé du facteur d'inertie) est supérieur ou égal à l'angle de consigne, alors on envoie une impulsion dans le sens contraire à la rotation pour freiner les moteurs puis on les éteint.

2.4.4 - Tests de validation

La méthode de test est similaire à celle employée pour le déplacement rectiligne. Nous avons dans un premier temps évalué la correspondance entre l'angle de rotation donnée en commande et celui de la rotation réellement effectuée par le robot visuellement. Dans un second temps, pour plus de précision, nous avons marqué l'orientation de départ et d'arrivée du robot au sol et avons utilisé un rapporteur pour mesurer l'angle correspondant. Ceci nous permet d'évaluer celui-ci avec une précision de l'ordre du degré, donc satisfaisante pour évaluer en fonction du critère imposé.

2.5 - Traitement des commandes envoyés de la station au robot

Pour pouvoir utiliser le blutooth, on appelle la bibliothèque "MOD24_NRF" developpé par Villou qui permet l'initialisation de la communication avec la fonction "initNRF24" et l'émission ou réception de donnée grâce aux fonctions .read et .write.

La station envoie systématiquement par bluetooth 3 valeurs à la carte Nucléo du robot. Les deux premières sont 0 ou 1 en fonction de si l'on effectue une rotation ou une translation et de leur sens, la dernière correspond à la valeur entrée dans l'interface. On utilise la fonction "Reception_NRF24" pour récuperer la valeur de translation ou de rotation qui avait été initialement choisit sur l'interface python. De plus, cette fonction appelle directement les fonctions précédemment explicitées "Avancer" et "Tourner" avec pour argument les valeurs de distance ou d'angle et un rapport cyclique arbitraire de 0.4.

Pour empêcher le robot de répéter successivement la même commande à cause d'erreur dans l'envoie ou la réception bluetooth, on oblige l'utilisateur à changer de valeur de commande pour chaque commande successive. C'est à dire qu'il n'est pas possible d'appliquer 2 fois la même commande successivement au robot.

On note aussi que les commandes sont effectués les une après les autres et qu'il n'est pas possible de tournet et d'avancer simultanement.

Fonction "reception_NRF24"

```
void Reception_NRF24(void)
{
    int A;  //Valeur d'angle de rotation
    int D;  //Valeur de distance à parcourir
    //int S;  //Sens soit de rotation, soit de translation
    char dataReceived[TRANSFER_SIZE]={0};
    /* 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);

        if (dataReceived[0]==0){  //0 correspond à la translation et 1 à la rotation
            //Récupère le sens de translation
            if (dataReceived[1]==1){  //1 correspond à avancer et 0 à reculer
                D=dataReceived[2];  //Récupère la valeur de translation avant
            }
            if (dataReceived[1]==0){
                D=-dataReceived[2];  //Récupère la valeur de translation arrière
            }
        }

        //Récupère le sens de rotation
        if (dataReceived[0]==1){
            if (dataReceived[1]==1){  //1 correspond à la gauche et 0 à la droite
                A=dataReceived[2];  //Récupère la valeur de rotation gauche
            }
            if (dataReceived[1]==0){
                A=-dataReceived[2];  //Récupère la valeur de rotation droit
            }
        }
        //On vérifie si le robot a reçu une nouvelle commande de distance
        if ((D!=0)&(D!=dist)){
            dist=D;
            Avancer(D, 0.4);
        }
        //On vérifie si le robot a reçu une nouvelle commande d'angle
        if ((A!=0)&(A!=angle)){
            angle=A;
            Tourner(A*1.0, 0.4);
        }

        // On affiche la distance ou l'angle de commande sur Teraterm
        debug_pc.printf("\tCommande = ");
        debug_pc.printf("D=%d \t",D);
        debug_pc.printf("A=%d \t",A);
        debug_pc.printf("\r\n");
    }
}
```

2.6 - Envoie des mesures du robot à la station

Pour envoyer les mesures du capteur de température et d'humidité du robot à la station, il faut d'abord choisir un capteur de température et d'humidité à cabler sur notre robot. On choisit, sans avoir le choix, le capteur **TEMP&HUM 14 CLICK** de chez MIKROE. Puis, on le cable à la carte nucléo sur notre robot. On note que l'on cable les pins SCL et SDA correspondant à la connexion I2C respectivement sur les broches A5 et A4 de la carte nucléo.

Schéma de cablage du module de mesure :

Notes	Pin	mikro BUS				Pin	Notes
	NC	1	AN	PWM	16	NC	
Reset	**RST**	2	RST	INT	15	NC	
	NC	3	CS	RX	14	NC	
	NC	4	SCK	TX	13	NC	
	NC	5	MISO	SCL	12	**SCL**	I2C Clock
	NC	6	MOSI	SDA	11	**SDA**	I2C Data
Power Supply	**3.3V**	7	3.3V	5V	10	**5V**	Power Supply
Ground	**GND**	8	GND	GND	9	**GND**	Ground

Dans le code, pour mesurer et envoyer la température et l'humidité, on utilise la fonction "Emission_NRF24". Celle-ci commence par lire les valeurs de température et d'humidité grâce à la fonction "readTRH" présente dans la bibliothèque "TEMPHUM_14_CLICK" développé par Villou. Cette fonction passe par un protocole I2C préalablement initialisé pour récupérer les mesures du capteur. Puis, elle envoie les octets correspondant à la température et à l'humidité par bluetooth.

Fonction "Emission_NRF24"

```
void Emission_NRF24(void)
{
    float TempToSend;
    float HumToSend;

    /*Lecture des valeurs de température et d'humidité*/
    my_sensor.readTRH(&TempToSend, &HumToSend);
    printf("T=%f / H=%f \r\n\n ", TempToSend, HumToSend);

    /*On suppose que T est compris entre 10°C et 30°C et que H entre 30 et 50.*/
    char T = (TempToSend-10)*255.0/20;
    char H = (HumToSend-30)*255.0/20;
    char dataToSend[TRANSFER_SIZE]={T, H, 0};

    /* Transmission donnée depuis nRF24 */
    nRF24_mod.setRfFrequency(2400);
    nRF24_mod.write( NRF24L01P_PIPE_P0, dataToSend, TRANSFER_SIZE );
    //debug_pc.printf( "SENT\r\n");
    wait_us(100000);
}
```

Lors du transfert bluetooth des mesures, on code la température et l'humidité sur un octet respectif, c'est à dire 255 valeurs. Afin d'assurer une certaine précision de mesure, on suppose que les mesures de températures seront comprises entre 10° et 30° et que les mesures d'humidité seront comprises entre 30 et 50. Cela est raisonnable dans les conditions du laboratoire mais si ce robot était vraiment envoyer sur Mars, il faudrait envoyer plus d'octet pour pouvoir mesurer une plus grande plage de température et d'humidité avec une bonne précision.

Ainsi, dans le code du robot, on multiplie la température par $(T - 10^\circ) \times \frac{255}{30^\circ - 10^\circ}$ pour être compris entre 0 et 255. On fait de même avec l'humidité en multipliant par $(H - 30) \times \frac{255}{50-30}$.

Enfin, dans le code de la station, on multiplie la température par $T \times \frac{30^{\circ}-10^{\circ}}{255} + 10^{\circ}$ pour retrouver la valeur de température mesurer. On fait de même avec l'humidité en multipliant par $H \times \frac{50-30}{255} + 30$. La précision sur les mesures est de $\frac{X_{max}-X_{min}}{2^8-1}$.

2.7 - Robot

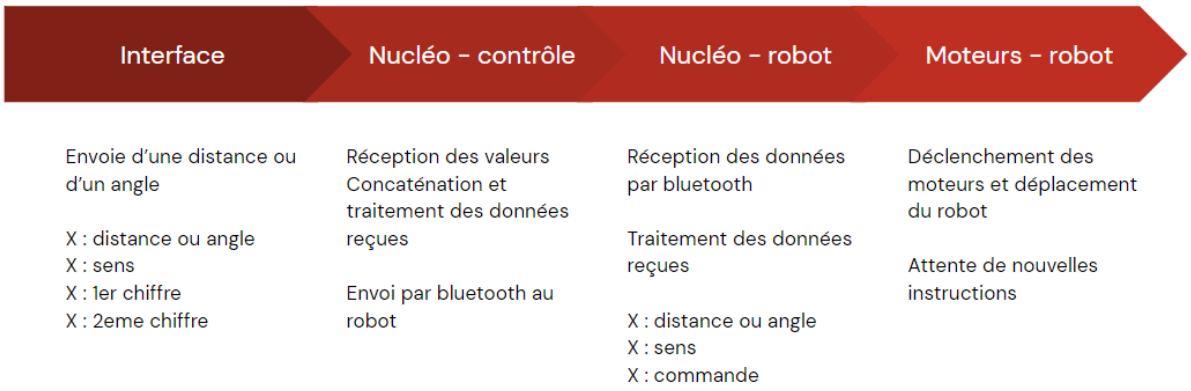
2.7.1 - Fonctionnement général

Au moment de l'assemblage des différentes parties, nous avions préparé des bibliothèques avec toutes nos fonctions exceptés "Emission_NRF24" et "Reception_NRF24". Cependant, il y avait des erreurs alors nous avons fait un unique fichier .cpp contenant toutes les fonctions du robot que nous avions écrites.

Nous avons d'abord essayé de mettre sur Ticker les fonctions `Emission_NRF24` et `Reception_NRF24` permettant ainsi de recevoir les commandes et d'envoyer les mesures à intervalle régulier. Cependant, cella n'a pas fonctionné, peut être à cause des quelques delais nécessaire à la mesure dans les fonction de la bibiliothèque `TEMP&HUM14 CLICK`.

Alors, nous avons initialisé la communication bluetooth grâce à la fonction `initNRF24` développée par Villou puis la communication i2c grâce à la fonction `initMesure` developpée par Villou dans le `main`. Puis, nous appelons les fonctions `Emission_NRF24` et `Reception_NRF24` en boucle dans le `while` du main.

Ainsi, grâce à la fonction `Reception_NRF24`, le robot répond aux commandes dès qu'il les reçoit par bluetooth et grâce à la fonction `Emission_NRF24`, le robot envoie par bluetooth et en boucle les mesures à la station. Malheureusement, l'envoi des mesures ne fonctionne plus lorsqu'on l'ajoute à toutes les autres fonctions nécessaires au fonctionnement du robot.



2.7.2 - Code du robot

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/9bd3d4e8-0d52-480b-86d3-482569b93d4d/Robot.cpp

▼ Bibliothèque "MOD24_NRF"

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/b7146d3d-b31c-44b2-a190-feb1a1606d9e/MOD24_NRF.cpp

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/fafccc96-3247-494f-bcde-f5a5468c70f5/MOD24_NRF.h

▼ Bibliothèque “TEMP&HUM14 CLICK”

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/81b28da6-e844-4efd-82d8-d555c5859279/TEMPHUM_14_CLICK.cpp

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/bee4ae6c-6495-453a-9ae0-8ee183c514f2/TEMPHUM_14_CLICK.h

3 - Bilan

3.1 - Partie technique / Avancement final

Comme indiqué et montré lors de la présentation au jury, le prototype remplit plusieurs des critères et exigences fournies. Il est capable de se déplacer en ligne droite et de tourner sur lui même selon une commande fournie à distance par un ordinateur connecté à un module bluetooth.

Ces déplacements sont néanmoins relativement imprécis; les **exigences de fiabilité** (erreur de 2 cm sur la position et 3° sur l’angle) de sont **pas satisfaites**. Plus de temps aurait été nécessaire afin de mettre en place un asservissement efficace. Le **critère de rapidité** est en revanche **respecté** (en n’utilisant que 40% de la puissance des moteurs).

Nous avons manqué de temps pour étudier l’autonomie de notre robot, et ne savons donc pas si il est capable de parcoutir 1 km avant de devoir recharger. Enfin, le critère d’ergonomie est lui aussi respecté ; l’interface est simple est utilisable sans connaissance préalable.

Notons cependant que certaines fonctionnalités ne sont pas complètement opérationnelles : le robot n’est pas capable de transmettre les mesures de ses capteurs à l’ordinateur de contrôle en raison de problèmes techniques avec la liason bluetooth (difficultés à transmettre les mesures et recevoir les commandes simultanément).

Enfin, la fonctionnalité d’affichage des mesures du robot en fonction de la distance ou du temps n’a pas été mise en place, faute de temps

3.2 - Retour d'expérience de l'équipe (comparaison Gantt...)

En termes d’organisation, il est difficile d’établir une comparaison entre le déroulement initialement prévu et la réalité du projet. Nos premières estimations portaient sur quelques séances, mais elles ont vite été déraillées à cause de problèmes de matériel (changement de carte par exemple) qui nous ont parfois forcé à passer une séance entière pour revenir au point ou nous nous étions arrêtés lors de la séance précedente. Nous n’avons donc pas pu continuer a essayer de faire des prédictions sur notre avancement au delà d’une séance ou deux.

Néanmoins, voici le diagramme de Gantt final, tenu à jour séance par séance. La distinction entre les deux équipes de travail y est claire. Notons également que n’y figurent pas quelques heures de travail

supplémentaires fournies afin de compnser le temps perdu sur certaines séances, et qui ont été cruciales pour pouvoir présenter un produit fonctionnel.