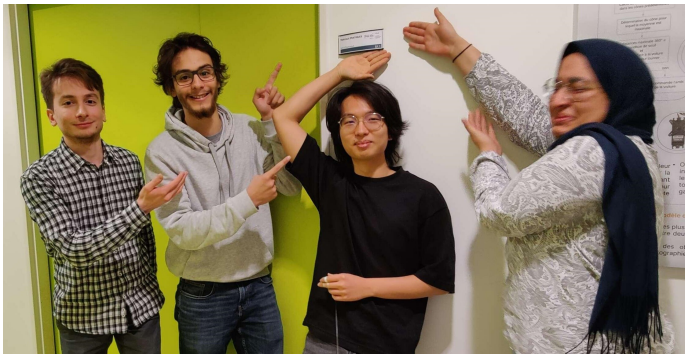
Rapport Technique : PROTIS
VéroniCar

Salma Elmiz, Yi Li, Jules Amico, Niels Guldner

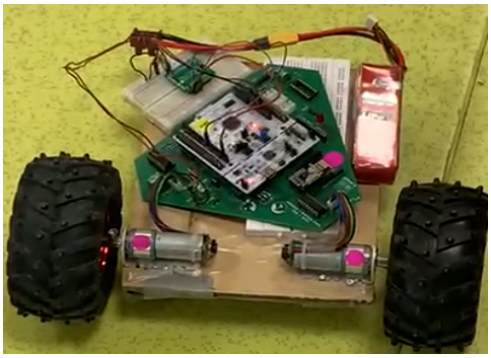
13 avril 2023

Résumé

Nous souhaitons développer un robot autonome guidé à distance permettant d'explorer le sol particulier de Mars, pour détecter la présence d'eau.



(a) L'équipe VéroniCar



(b) VéroniCar

FIGURE 1 – L'équipe VéroniCar et VéroniCar

Table des matières

1 Introduction

1.1 Objectifs du projet

Nous avons pour objectif la conception d'un robot guidé à distance permettant de détecter l'humidité et la température du sol qu'il explore. Le robot sera capable d'effectuer des trajets composés de lignes droites et de rotations. La commande du robot sera effectuée depuis un ordinateur et une interface homme-machine, via laquelle on donnera des instructions à l'avance. Chaque déplacement du robot enverra de nouvelles données d'humidité et de température au poste de contrôle.

1.2 Schéma de principe

Dans la figure 2, nous présentons le principe de fonctionnement simplifié du robot.

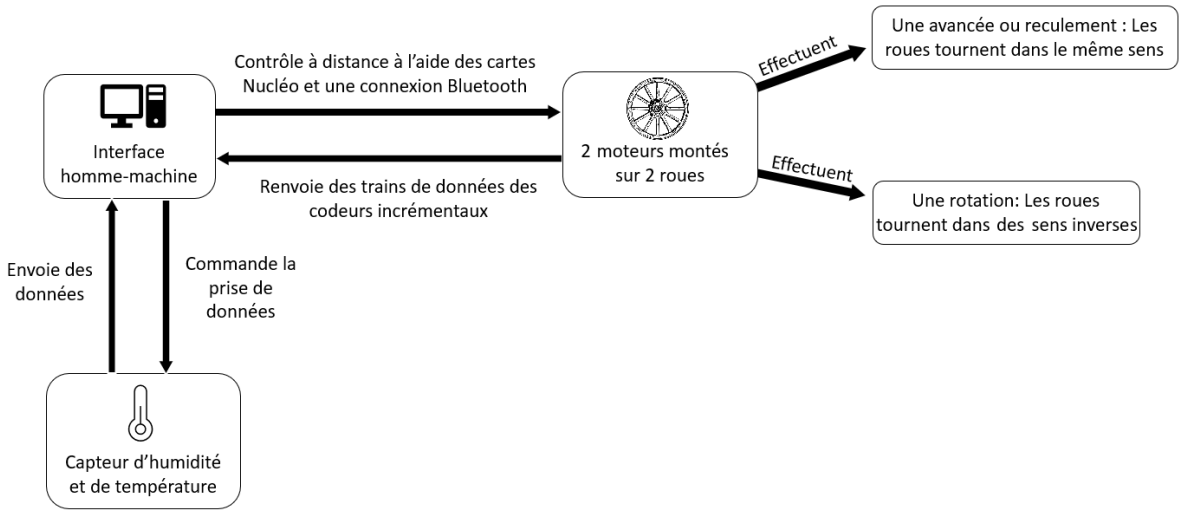


FIGURE 2 – Schéma de principe

1.3 Matériel utilisé

Ici, nous détaillons chaque modèle de composant utilisé, et nous joignons les documentations techniques lorsqu'elles existent. Certains composants ont été modifiés par le LEnsE dans le cadre du projet : dans ce cas, nous donnons la documentation nécessaire à leur compréhension.

- Châssis de voiture du LEnsE
  - Carte de circuit triangulaire (figures 3 et 6)
  - Carte `Nucléo-L476RG`
    - https://docs.rs-online.com/2f7f/0900766b81440071.pdf
  - 2 roues motrices
  - 2 moteurs à courant continu `IG220019X00015R`
    - https://digilent.com/reference/_media/motor_gearbox/290-006_ig220019x00015r_ds.pdf
  - 2 codeurs incrémentaux montés sur les moteurs : `360 CPR`
  - 1 roue libre
  - 2 ponts en H : `L293DNF`
    - https://www.ti.com/lit/ds/symlink/l293d.pdf?ts=1680769202381&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FL293D
- 2 modules Bluetooth `RF NRF24L01`
  - https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf
- 1 batterie LiPo : `Conrad energy 1344149 25 C` (11.1 V et 1800 mAh)
- Capteur d'humidité et de température : `Temp&Hum 14 Click MIKROE-4306`
  - https://docs.rs-online.com/d3d8/A700000009001611.pdf

1.4 Cahier des charges

Le robot doit pouvoir :
— se déplacer en ligne droite, selon une distance transmise depuis un système distant,
— effectuer des rotations sur lui-même d'un angle transmis depuis un système distant,
— transmettre des données d'humidité et de température au système distant.

Les performances attendues sont les suivantes :
— Rapidité : Le robot doit pouvoir avancer à une vitesse comprise entre 10 et 30 cm/s.

  Réalisation : La vitesse du robot n'a pas encore été étalonnée. Cependant, le robot est capable d'atteindre des vitesse supérieures à 30 cm/s. Avec une petite séance d'étalonnage, le robot pourra avancer à une vitesse définie comprise entre 10 et 30 cm/s.
— Fiabilité sur la position : Une erreur maximale de 2 cm est tolérée sur la position.

  Réalisation : À basse vitesse, le poids total du robot rend l'effet de l'inertie négligeable. L'erreur se reporte donc sur le train de données du codeur incrémental. D'après nos tests de fonctionnement, il y a une erreur de ± 1 front montant. Cela correspond à une erreur algébrique sur la position de ± 0.2 cm. Cependant, en pratique, nous avons plutôt une erreur de l'ordre de centimètre (l'effet d'inertie n'est pas totalement négligeable). Le critère est donc plus ou moins validé.
— Fiabilité sur l'angle : Une erreur maximale de 3° est tolérée sur l'angle.

  Réalisation : De la même manière que l'erreur sur la position, l'erreur sur l'angle se reporte sur le train de données du codeur incrémental. Une erreur de ± 1 front montant correspond à une erreur algébrique sur l'angle de $\pm 3.14°$. Cette performance peut être améliorée, mais elle reste raisonnable.
— Autonomie : La batterie doit avoir une autonomie suffisante pour que le robot puisse parcourir 1 km.
  Réalisation : Nous utilisons une batterie de capacité 1800 mA h. Le robot pèse au total environ 2 kg, donc nous pouvons supposer que chaque roue supporte 1 kg. D'après la documentation du moteur `IG220019X00015R`, on estime que le courant d'utilisation vaut 1 A à 1 kg de charge du moteur. Cela correspond à une autonomie d'environ 54 minutes pour un moteur. Avec une vitesse maximale de 30 cm/s, le robot peut ainsi parcourir environ 972 m sur une charge complète de la batterie. Ainsi, le critère d'autonomie est relativement bien validé.
— Ergonomie : L'interface homme-machine, permettant de transmettre les ordres de parcours, doit pouvoir être utilisée facilement sans formation préalable. Les données doivent être affichées en fonction de la distance ou du temps.
  Réalisation : Une interface compréhensible et simple a été conçue à travers Spyder.

1.5 Notice d'utilisation

1.5.1 Câblage de la voiture :

Les moteurs et la Nucléo sont liés à travers une carte de circuit triangulaire réalisée par le LEnsE (figure 3).

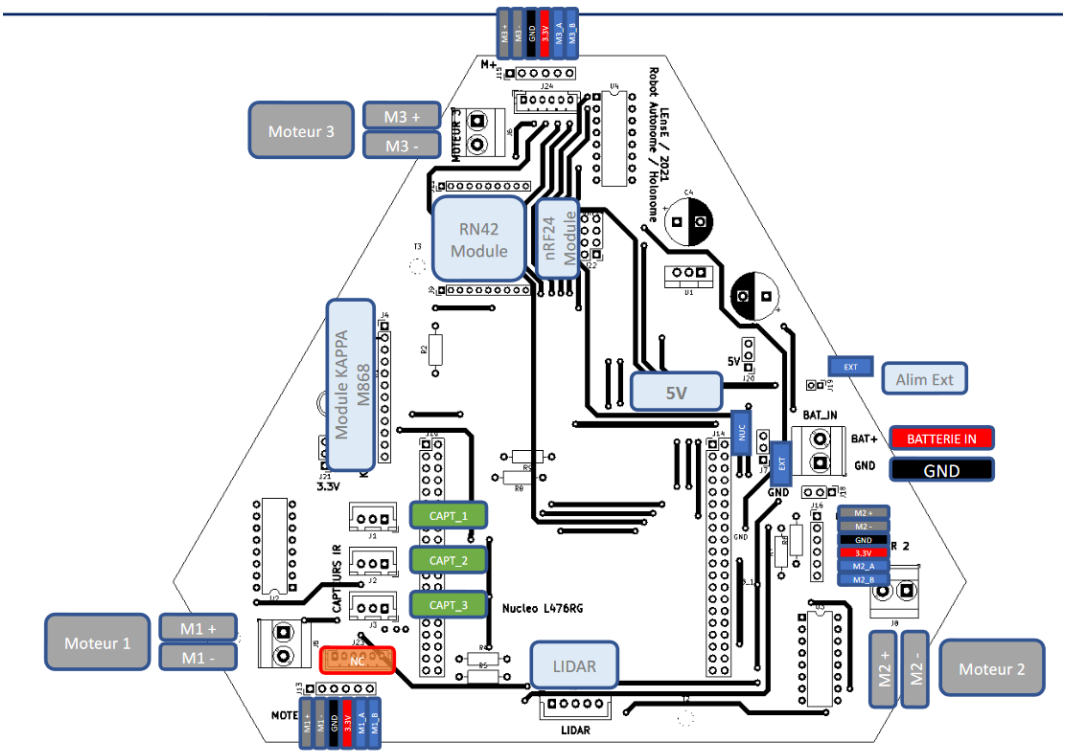


FIGURE 3 – Brochages du circuit de la carte triangulaire (LEnsE)

Dans notre utilisation, il faut connecter une batterie de tension nominale 12 V à l'alimentation de la carte (`BATTERIE IN` et `GND`). De plus, pour alimenter la carte Nucléo avec la batterie externe, il faut mettre le cavalier de la carte sur la position externe (`EXT`). Sur la figure 1b, nous montrons la voiture correctement montée et câblée.

1.5.2 Instruction d'utilisation de l'interface homme-machine

L'interface homme-machine se présente comme à la figure 4. L'utilisateur est invité à y renseigner la valeur, signée, de la distance à parcourir, suivie de l'angle de rotation. Par exemple, pour ordonner une rotation de 5° dans le sens horaire et un déplacement de 35 cm, on écrira l'instruction donné dans le tableau 1 :

distance			angle		
signe	valeur		signe	valeur	
+	3	5	-	0	5

TABLE 1 – Exemple des caractères à entrer sur l'ordinateur. La distance est exprimée en cm et l'angle en degré. Un angle positif correspond au sens trigonométrique.

Ces informations seront transmises et effectuées par le robot. Une fois le déplacement réalisé, le robot renvoie la température et l'humidité au poste de contrôle, qui affiche ces données à l'utilisateur (par manque de temps et par priorité au bon fonctionnement de l'ensemble du prototype, nous n'avons affiché que la température ; cependant, il très aisé de rajouter l'humidité en suivant le même procédé). Une fois ces données affichées, l'utilisateur est de nouveau invité à entrer une commande de déplacement.

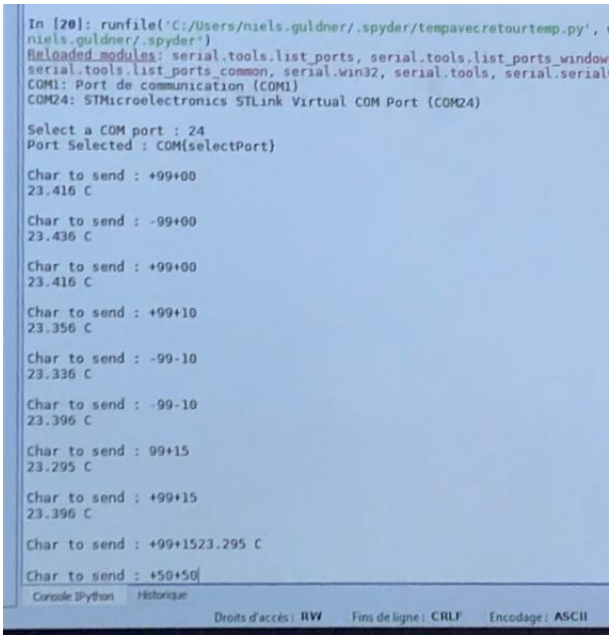


FIGURE 4 – Capture d'écran de l'interface homme-machine sur l'ordinateur dans le poste de contrôle. Un scripte Python est utilisé dans l'environnement Spyder.

1.6 Schéma fonctionnel

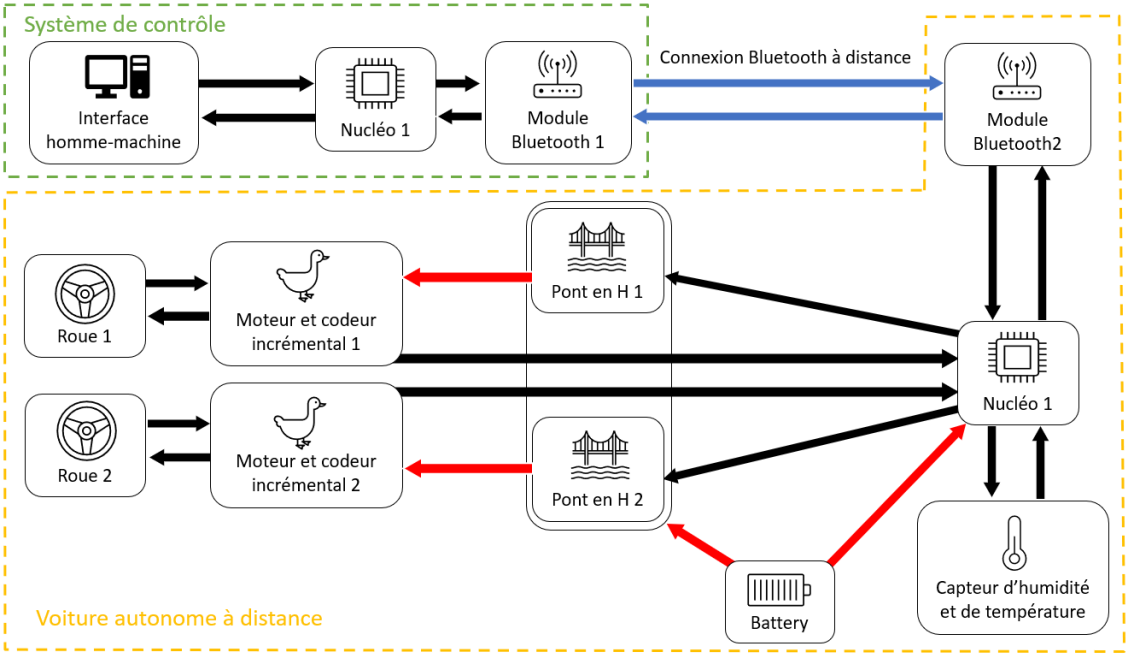


FIGURE 5 – Schéma fonctionnel du projet

2 Description des fonctions

2.1 Commande des moteurs

Les ports de connexion entre la carte de circuit triangulaire et la carte Nucléo sont donnés figure 6 :

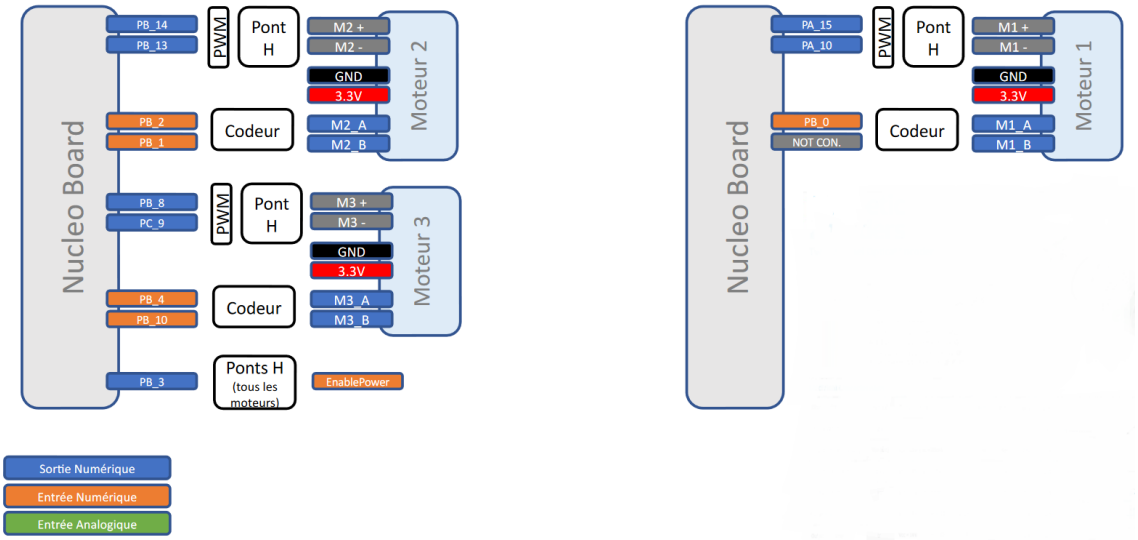


FIGURE 6 – Ports de connexion entre la carte de circuit triangulaire et la carte Nucléo (LEnsE)

Dans ce projet, nous utilisons les ponts en H pour les moteurs 2 et 3 sur la carte triangulaire. Le principe de fonctionnement est le suivant : la Nucléo envoie une commande en tension de pulse-width modulation (PWM) au pont en H. En ajustant le rapport cyclique de la PWM, l'utilisateur ajuste la puissance délivrée au pont en H. Le principe de la PWM est rappelé figure 7.

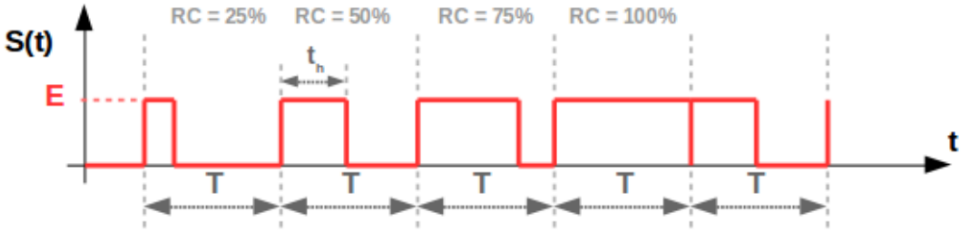


FIGURE 7 – Principe du PWM (LEnsE)

Le pont en H amplifie le signal de tension reçu et délivre un courant continu au moteur. Ce courant est proportionnel à l'énergie reçue par le pont en H. Ainsi, le rapport cyclique du PWM détermine la vitesse de rotation des moteurs.

Quand les moteurs tournent, leurs codeurs incrémentaux renvoient des signaux à la carte Nucléo. Ces signaux se présentent sous forme d'un train de données rectangulaires (figure 8). Chaque front montant correspond à 1 incrément du codeur, et donc à un certain degré de rotation du moteur. Notre codeur incrémental a une deuxième voie, utile pour connaître le sens de rotation du moteur. Cependant, dans notre projet, la première voie est suffisante.

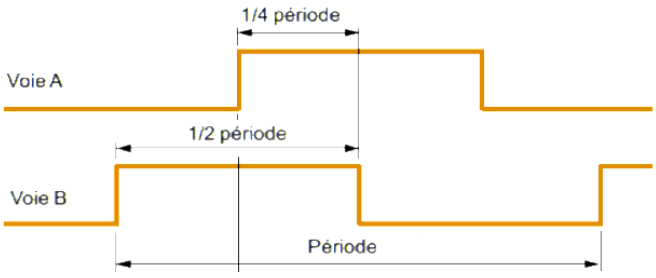


FIGURE 8 – Signaux envoyés par le codeur incrémental

Nous donnons dans la table 2 les ports de connexion et leur utilisation.

Nucléo	Moteur	Utilisation
PB_14 (Sortie analogique PWM)	M2+	Moteur 2 avancer
PB_13 (Sortie analogique PWM)	M2-	Moteur 2 reculer
PB_8 (Sortie analogique PWM)	M3+	Moteur 3 avancer
PC_9 (Sortie analogique PWM)	M3-	Moteur 3 reculer
PB_2 (Entrée analogique)	M2_A	Moteur 2 codeur incrémental voie A
PB_1 (Entrée analogique)	M2_B	Moteur 2 codeur incrémental voie B
PB_4 (Entrée analogique)	M3_A	Moteur 3 codeur incrémental voie A
PB_10 (Entrée analogique)	M3_B	Moteur 3 codeur incrémental voie B

TABLE 2 – Les ports de connexion de la carte de circuit triangulaire et leur utilisation

Avant toute opération, il faut étalonner le codeur incrémental. Le protocole est le suivant : on fait tourner le moteur à basse vitesse, et on chronomètre jusqu'à ce que la roue ait fait 10 tours. En faisant une moyennage sur 10 périodes, nous lisons sur l'oscilloscope la période du codeur incrémental. En répétant plusieurs fois l'expérience, on détermine le lien entre les fronts montants du signal du codeur et la rotation de la roue sur elle même :

$$1 \text{ front montant} \longleftrightarrow \text{rotation de la roue de } 6^\circ$$

En mesurant les dimensions de la voiture, nous trouvons que le rayon des roues est $R_{\text{roue}} \approx 6.5$ cm et que la distance entre les deux roues est $D_{2\text{ roues}} \approx 24.8$ cm. Ainsi, nous en déduisons la distance parcourue par la voiture pendant 1 front montant (les deux roues tournant dans le même sens) :

$$1 \text{ front montant} \longleftrightarrow 0.72 \text{ cm d'avancement}$$

De même, nous trouvons la rotation effectuée par la voiture pendant 1 front montant (les deux roues tournant dans des sens opposés) :

$$1 \text{ front montant} \longleftrightarrow 3.14\ ^\circ \text{ de rotation de la voiture}$$

Nous définissons dans nos fonctions un compteur qui compte le nombre de tics, c'est à dire de changements de signal du codeur incrémental (de 0 à 1 et de 1 à 0). Ce compteur représente donc le double du nombre de front montants. Dans notre programme, nous avons défini des fonctions permettant de faire de faire bouger la voiture de la distance ou l'angle souhaité, en utilisant une boucle `while` qui s'arrête lorsque le bon nombre de tics est atteint.

Cependant, il y a un problème intrinsèque à nos fonctions : la fréquence d'échantillonage du signal du codeur incrémental n'est pas définie. En effet, dans nos fonctions, le comptage est séquentiel et peut être ralenti ou perturbé par les autres éléments du code (un `printf` par exemple).

Par manque de temps, et beaucoup trop perturbés par les problèmes techniques de matériel, nous n'avons pas réalisé de comptage par interruption, ce qui aurait pu résoudre notre problème. Nous aurions également pu mettre en place une boucle de rétroaction pour plus de fiabilité sur la distance et l'angle.

2.2 Interface Homme-Machine

L'interface homme-machine, illustrée figure 4, est réalisée à travers un script Python sur Spyder. L'objectif de ce programme est de prendre les données qu'entre l'utilisateur sur l'ordinateur, et de les envoyer à la carte Nucléo, branchée en USB. Pour cela, il faut adapter le programme sur Python pour récupérer les données et les transmettre à la carte, ainsi qu'adapter celui sur la carte Nucléo pour recevoir et traiter les données.

Le script Python est retranscrit ci-dessous. Il est basé sur celui de Julien Villemejane sur le site du LEnsE, et adapté à nos besoins et à Spyder version 2.7 (il fallait changer les `imput` en `raw_imput` qui fournissent des `string` en sortie, et les traiter correctement). Le code attend une donnée entrée par l'utilisateur, et l'envoie directement à la Nucléo. La valeur `q` permet de quitter le programme. Une fois les données transmises à la carte Nucléo, le programme attend que celle-ci lui envoie les données de la température mesurée (sur 8 bytes). Une fois ces données reçues, le programme les affiche et invite de nouveau l'utilisateur à entrer une commande.

```python
from serial import Serial
#import time
import serial.tools.list_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))
    # To select the port to use
    selectPort = input("Select a COM port : ")
    print("Port Selected : COM{selectPort}")
    # To open the serial communication at a specific baudrate
    serNuc = Serial('COM'+str(selectPort), 115200)  # Under Windows
       only
    appOk = 1
    while appOk:
        data_to_send = raw_input("Char to send : ")
        if data_to_send == "q" or data_to_send == "Q":
            appOk = 0
        else:
            # serNuc.write(bytes(data_to_send, 'ascii'))
            serNuc.write(data_to_send)
            while serNuc.inWaiting() == 0:
                pass
            #time.sleep(0.5)
            data_rec = serNuc.read(8)  # bytes
            print(str(data_rec))

    serNuc.close()
```

2.3 Communication Radio

La communication radio s'effectue à l'aide du module `nRF24` placé sur la carte triangulaire (voir figure 3). Ce module est, de ceux que nous avons essayés, le seul qui fonctionne correctement avec notre matériel et avec Keil Studio (la nouvelle version de Mbed). Pour piloter le module, nous sommes partis du code fourni par Julien Villemejane. Ce code permettait d'envoyer une trame de 8 octets à l'appui du bouton de la Nucléo. Nous avons entre autre modifié l'adresse, pour ne pas interférer avec les modules des autres groupes PROTIS travaillant sur des systèmes de communication sans fils.

2.3.1 Émission de la commande et réception de la température

La carte Nucléo liée à l'ordinateur reçoit la commande par le code Python. Pour cela, une série de six fonctions `getchar` permet de recevoir les données. La boucle `while` en amont permet d'attendre que les données soient bien reçues par la Nucléo. En effet, ces différents composants sont indépendants et travaillent donc chacun à leur vitesse. Ainsi, il est nécessaire de traiter les données de façon à s'assurer que les informations soient bien communiquées.

Une fois la trame d'information récupérée, on vient les transmettre dans le bon canal à l'autre Nucléo.

```
// Fonction transmet distance et rotation du module BT nRF24L01
void transmettreConsigne(int *decideur){
    char data1;
```

```
    //mettre un decideur qui switch entre read et write
       alternativement
    /* Lecture donn e depuis nRF24 */
    if ( *decideur == 1 && nRF24_mod.readable() ) {

        // ...read the data into the receive buffer
        rxDataCnt = nRF24_mod.read( NRF24L01P_PIPE_P0,
           dataReceivedConsigne, 8);

        // Display the receive buffer contents via the host serial
           link
        for ( int i = 0; i < rxDataCnt; i++ ) {
            debug_pc.printf("%c", dataReceivedConsigne[i]);
        }

        // on  coute  desormais le pc
        *decideur = 0;
    }

    //  coute  le pc
    if(*decideur == 0) {
        // on r cupere les 6 caract res de l'ordi fourni de python
        while ( not debug_pc.readable() ) {}
        data1=debug_pc.getc();
        distanceAngle[0]=data1;
        while ( not debug_pc.readable() ) {}
        data1=debug_pc.getc();
        distanceAngle[1]=data1;
        while ( not debug_pc.readable() ) {}
        data1=debug_pc.getc();
        distanceAngle[2]=data1;
        while ( not debug_pc.readable() ) {}
        data1=debug_pc.getc();
        distanceAngle[3]=data1;
        while ( not debug_pc.readable() ) {}
        data1=debug_pc.getc();
        distanceAngle[4]=data1;
        while ( not debug_pc.readable() ) {}
        data1=debug_pc.getc();
        distanceAngle[5]=data1;

        nRF24_mod.setRfFrequency(2400);
        nRF24_mod.write( NRF24L01P_PIPE_P0, distanceAngle,
           TRANSFER_SIZE_CONSIGNE );
        wait_us(100000);

        // on ecoute la Nucleo
        *decideur = 1;
    }
}
```

Une autre partie du programme est conçue pour renvoyer la température à l'interface Python. La gestion d'une partie est effectuée avec un switch avec la variable `decideur` qui vient alterner entre l'envoi de la commande et la communication de la réponse. Enfin, l'interface Python affiche la température.

2.3.2 Réception de la commande et émission de la température

La réception de la commande est différente de celle de la partie précédente, puisqu'il est nécessaire de convertir la chaîne de caractères en `int`. On traite alors les caractères un par un et on ajoute le signe à partir du codage ASCII. La fonction modifie directement les variables en entrées de la fonction avec les pointeurs sur `distance` et `angle`. Ainsi dans le `main` on peut directement les utiliser pour commander les roues. Un compteur `c` est aussi utilisé sous forme de pointeur pour garder en mémoire si une nouvelle trame de donnée a été reçue ou non.

```
// Fonction transmet distance et rotation du module BT nRF24L01
void transmettreConsigne(int *distance, int *angle, int *c){
    /* Lecture donn e depuis nRF24 */
    if ( nRF24_mod.readable() ) {

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

        // Convert the receive buffer contents via the host serial
            link
        // convertit le code ASCII en nombre
        for ( int i = 0; i < rxDataCnt; i++ ) {
            dataConvert[i] = dataReceivedConsigne[i] - 48;
        }
        // - correspond    -3
        // + correspond    -5

        *distance = dataConvert[1]*10 + dataConvert[2];
        if (dataConvert[0] == -3)
        {
            *distance = *distance * -1;
        }

        *angle = dataConvert[4]*10 + dataConvert[5];
        if (dataConvert[3] == -3)
        {
            *angle = *angle * -1;
        }

        debug_pc.printf("distance : %d \r\n angle : %d \r\n", *
            distance, *angle);
        //debug sur teraterm

        *c=*c + 1;
        //incr le compteur pour savoir si on a recu de nouvelle
            donn e ou pas
    }
}
```

Une fois terminée la conversion des données de distance et d'angle, les moteurs effectuent leurs rotations, et la fonction `transmettreTempHum` est finalement exécutée pour renvoyer la température. La valeur de T est récupérée, puis convertie en chaîne de caractères avant d'être renvoyée avec le module radio. On ajoute l'unité pour indiquer qu'il s'agit de degrés Celsius.

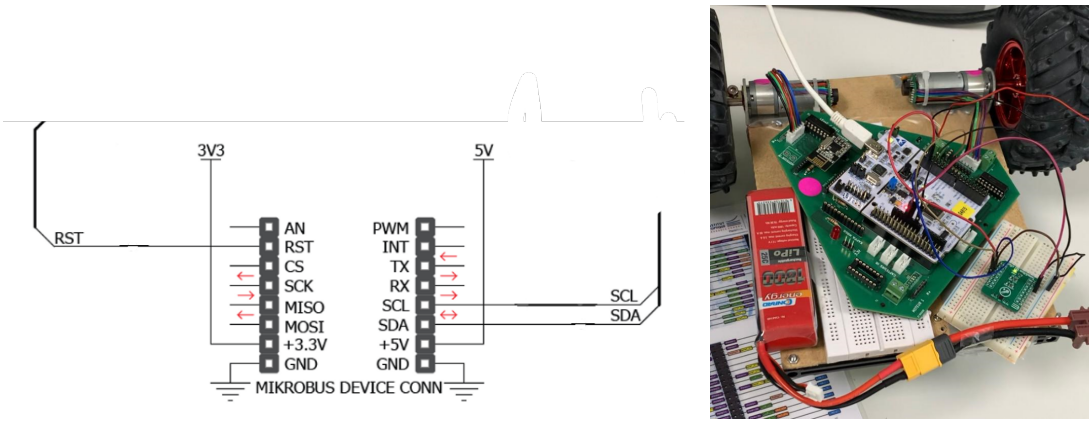
```
//faire fonction emission de temp et hum
void transmettreTempHum(float *T, float *H){

    char Tchar[8];
    int ret = snprintf(Tchar, sizeof Tchar, "%f", *T);
```

```
    // convertit le float en chaine de caract re

    Tchar[6]=' ';
    Tchar[7]='C'; //Afficher l'unit
    debug_pc.printf("%c%c%c%c%c%c%c%c \n", Tchar[0], Tchar[1], Tchar
        [2], Tchar[3], Tchar[4], Tchar[5], Tchar[6], Tchar[7]);
    // affiche pour le debug

    // envoi des donn es    l'autre carte
    nRF24_mod.setRfFrequency(2400);
    nRF24_mod.write( NRF24L01P_PIPE_P0, Tchar, 8 );
    debug_pc.printf( "SENT\r\n");
    wait_us(100000);
}
```

2.4 Détection de température et d'humidité

Le capteur que nous avons utilisé pour la détection de température et de l'humidité est `Temp&Hum 14 Click MIKROE-4306`. Vu les performances de ce capteur, il est sûrement inadéquat pour le climat martien. En effet, son intervalle de mesure de température va de -40 °C à 125 °C, tandis que la température sur Mars varie entre -143 °C et 27 °C. Nous l'utilisons néanmoins pour un premier prototype sur Terre.

En ce qui concerne la communication, ce capteur utilise un réseau de type `i2C`. Ainsi, il est nécessaire de choisir le bon câblage (cf. figure 9a), et d'assurer une bonne communication entre l'esclave (le capteur) et le maître (la carte Nucléo). Il ne faut pas brancher le port `Reset` (`RST`), sinon le capteur ne conserve jamais la valeur mesurée, et affichera la valeur minimale de -40 °C. Les ports essentiel au bon fonctionnement du capteur avec le code sont `SCL` et `SDA`, qui permettent la communication avec la Nucléo, ainsi que les pin `GND`, `+3.3V` et `+5.5V` (liaison directe avec les ports de la cartes).



(a) Câblage à réaliser (b) Câblage réalisé

FIGURE 9 – Câblage du capteur de température et d'humidité `Temp&Hum 14 Click MIKROE-430`

Les lignes de code et les bibliothèques nécessaires pour le programme de lecture ont été fournies par Julien Villemejane. Nous l'avons adapté pour pouvoir afficher les mesures sur l'IHM. Les valeurs obtenues sont ensuite envoyées, par une antenne bluetooth attachée au robot, vers l'antenne du centre de contrôle pour qu'elles soient affichées sur l'IHM.

Fonction de mesure et de lecture de la température et de l'humidité

```
void TempHum_14_Click::readTRH(float *temp, float *hum){
    // Conversion in fast mode
    cmd[0] = TEMPHUM_14_CLICK_CONV;
```

```
    ack1 = __i2c->write(TEMPHUM_14_CLICK_ADD << 1, cmd, 1);
    if(DEBUG_MODE)  printf("Conv Acq (W) = %d\r\n", ack1);
    thread_sleep_for(3);    // 3 ms
    // Read data
    cmd[0] = TEMPHUM_14_CLICK_READ_T_RH;
    ack1 = __i2c->write(TEMPHUM_14_CLICK_ADD << 1, cmd, 1);
    ack2 = __i2c->read(TEMPHUM_14_CLICK_ADD << 1, data, 6);
    int tEmp = (data[0] << 8) + (data[1]);
    int hUm = (data[0] << 8) + (data[1]);
    _temperature = -40.0 + 165.0 * tEmp / 65535;
    *temp = _temperature;
    _humidity = 100.0 * hUm / 65535;
    *hum = _humidity;
}
```

La fonction de lecture des données est composée de deux parties :

1. La première partie concerne la conversion et permet de s'assurer de la bonne lecture de la commande (si `ack1 = 0`, le capteur a bien reçu la commande). Le temps de repos de 3 ms est la période nécessaire pour effectuer la conversion.
2. La deuxième partie concerne la réalisation de la mesure, via l'envoi de la commande `write()` et son affichage par la commande `print()` a travers la ligne SDA. Les données de température et d'humidité sont ensuite stockées dans le tableau `data` de 6 octets. Ces mesures ne sont pas les valeurs réelles : une conversion numérique est nécessaire.

3 Bilan

3.1 Avancement final

En commençant ce projet, nous avions prévu l'ajout de plusieurs fonctionnalités, telles que l'arrêt automatique en cas d'obstacle, avec un capteur de distance placé à l'avant du robot. Nous voulions également ajouter une caméra sur le robot pour que la personne qui le contrôle soit capable de voir la route en temps réel. Nous n'avons pas eu le temps de mettre en place ces systèmes. Plusieurs imprévus ont été rencontré lors du projet, ce qui a perturbé l'avancement prévu. Malgré cela, les fonctions essentielles du projet ont réalisées.

Le planning des tâches que nous avons suivi pendant notre projet est représenté sur la figure 10.

Comme indiqué sur ce planning, les tâches suivantes du cahier des charges ont été réalisées :

— se déplacer en ligne droite, selon une distance donnée.
— effectuer des rotations d'un angle donné
— autonomie d'1 km de parcours
— vitesse comprise entre 10 et 30 cm/s.
— transmission de l'angle de rotation et de la distance par Bluetooth
— mesurer la température et de l'humidité
— une interface homme-machine facile d'utilisation

Les tâches qui restaient sont l'asservissement des moteurs pour avoir une erreur de 10 cm sur la position et 10° sur l'angle et la transmission de la température et de l'humidité chaque 10 minutes.
Ces dernières ne sont pas aussi compliquées pour mettre en place, c'était juste question de temps perdu dans d'autres problèmes comme :

1. un manque de documentation sur les codes et les câblages en ce qui concerne les tâches de communication,
2. de nombreux composants que nous avons grillé, ou qui ont grillé, semble t-il, d'eux mêmes.
3. un matériel présentant certains défauts : par exemple, une des roues avait besoin d'un petit coup pour se mettre en marche.

Planning

Table | Chronologie | + | Filtrer | Trier

Séance	Date	Person	Status
Documenter les matériels	31 janvier 2023 → 4 février 2023	Y	Done
Schéma fonctionnel	31 janvier 2023 → 4 février 2023	jules	Done
Communication Bluetooth	8 février 2023 → 8 mars 2023	Niels, Salma EL MIZ	Done
Gestion des moteurs (position et angle)	8 février 2023 → 14 mars 2023	Y, jules	Done
Récupérer des données du capteur température et humidité	8 mars 2023 → 15 mars 2023	Salma EL MIZ	Done
Interface de contrôle sur ordinateur	8 mars 2023 → 15 mars 2023	Niels	Done
Asservissement en position et en angle			Not started
Raccordement moteur bluetooth	22 mars 2023 → 29 mars 2023	Niels, Y, jules	Done
Étalonnage des codeurs incrémentale			Not started
Fiche documentation		Y, Niels, jules, Salma EL MIZ	Done
Présentation	29 mars 2023	Salma EL MIZ	Done

FIGURE 10 – Planning suivi lors de notre projet

Conclusion sur la partie technique du projet

Lors des répartitions des tâches et des discussions sur les fonctionnalités du robot, nous avons décidé que notre projet serait réussi techniquement si les grandes fonctions du cahier des charges étaient remplies, et si nous étions capables de comprendre le fonctionnement et les codes utilisés. Cela a été réalisé avec succès. Nous sommes donc fiers de dire que notre projet a été réussi.

3.2 Retour d'expérience de l'équipe

Ce projet nous a permis de développer des compétences en gestion de projet, du point de vue du travail d'équipe et de la communication, comme la mise en place d'un diagramme de Gant. En travaillant en équipe, nous avons collaboré pour définir les objectifs, organiser les tâches et définir des échéances. Quant aux problèmes techniques, bien qu'ils nous aient ralentis, ils nous ont été instructifs : nous avons appris de nos erreurs (par exemple, l'importance de mesurer une tension avec précaution, afin de ne pas causer de court-circuit). En travaillant en équipe, nous avons su identifier chacun des problèmes et avons pu les résoudre.