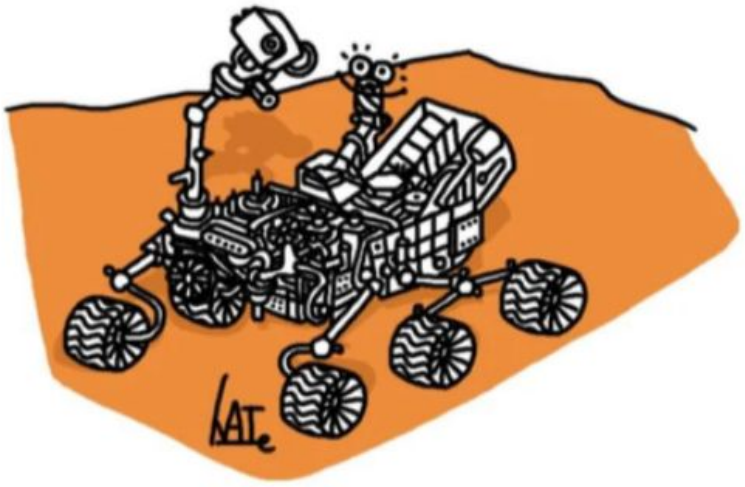
Projet Veronica



SOMMAIRE

1. Introduction

Nous souhaitons construire un robot à destination de Mars. Ce robot devra recevoir des commandes depuis la Terre pour se déplacer rectilignement et angulairement. Il devra également renvoyer à la Terre des informations concernant l'atmosphère martienne. Tout d'abord nous étudierons l'envoi des commandes et la réception des informations via la liaison série de type RS-232. Ces commandes seront envoyées grâce à une interface graphique intuitive. Une fois les commandes envoyées, elles sont reçues par le robot. Il faut alors agir sur la motorisation des roues pour se déplacer à la position demandée par l'utilisateur. Il faudra vérifier via un asservissement le respect de ces consignes. Finalement, le robot envoie des informations sur la température et l'humidité martienne. Nous nous intéresserons donc au fonctionnement de capteurs adéquats.

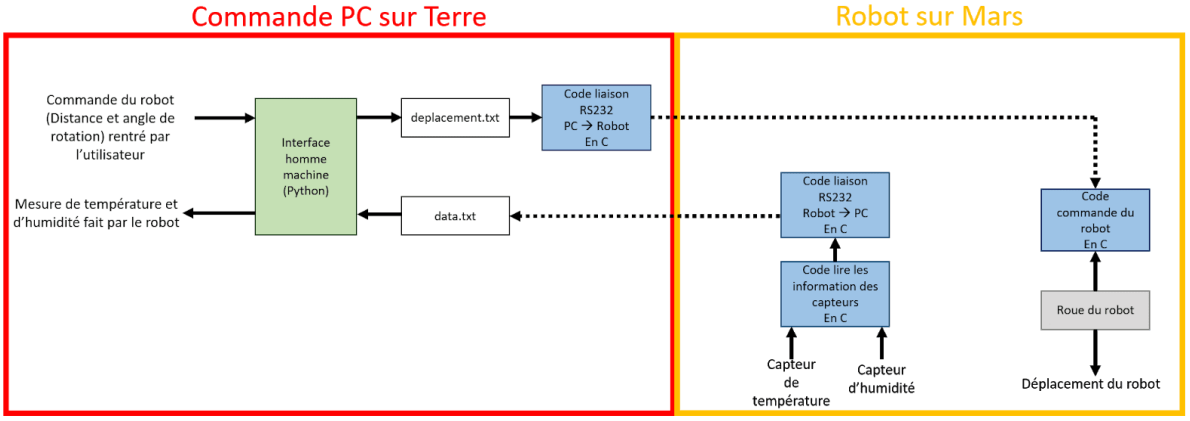


Schéma fonctionnel du robot Véronica

2. Cahiers des charges

La société Solec nous a fixé des objectifs à respecter. Ces contraintes nous permettent d'assurer le bon asservissement du robot. Ces critères sont rappelés ci-dessous.

Rapidité : Le robot doit pouvoir avancer à une vitesse comprise entre 10 et 30 cm/s.
Fiabilité : Une erreur maximale de 2 cm est tolérée sur la position. Une erreur maximale de 3° est tolérée sur l'angle.
Autonomie : Le robot doit pouvoir réaliser un parcours de 1km sans que ses batteries ne soient rechargées.

Ergonomie : L'interface Humain-Machine, permettant de transmettre les ordres de parcours, doit pouvoir être utilisée sans formation préalable. Les données doivent pouvoir être affichées en fonction de la distance ou du temps.

3. Interface graphique

Pour que l'utilisateur puisse contrôler le robot à distance et recevoir les informations de température et d'humidité nous avons décidé de créer une interface graphique sous Python en utilisant Tkinter. Avec cette interface, l'utilisateur pourra commander le robot en envoyant une valeur de distance à parcourir et un angle (Pour que le robot tourne). De plus on veut recevoir comme information la température et le taux d'humidité de Mars, on affichera donc sur une graphique les valeurs renvoyées par le robot.

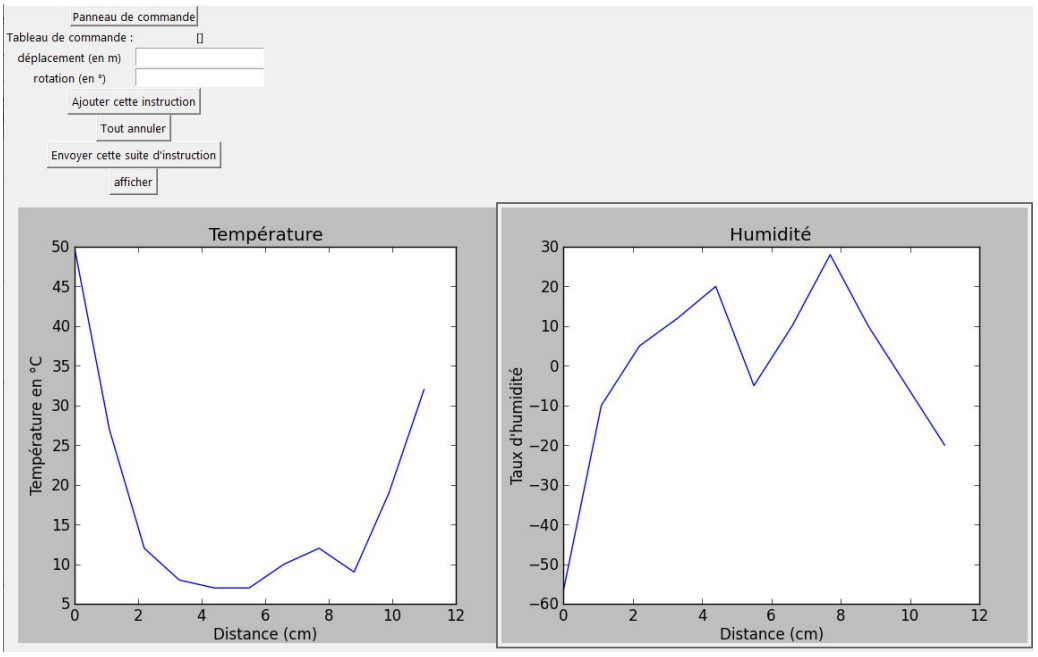


Figure : Apparence final de l'interface graphique

En haut on retrouve l'interface de commande, qui va servir à l'utilisateur pour marquer ces instructions au robot et décider d'afficher les valeurs lues par le capteur.

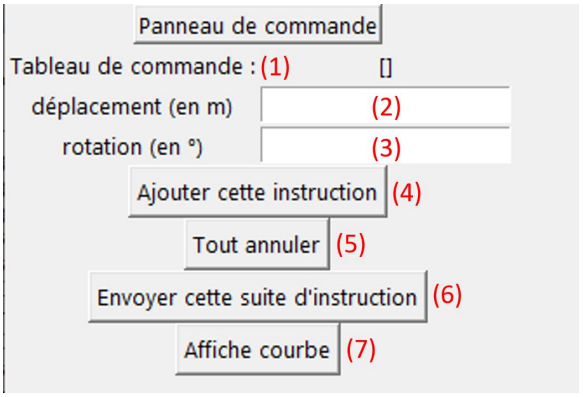
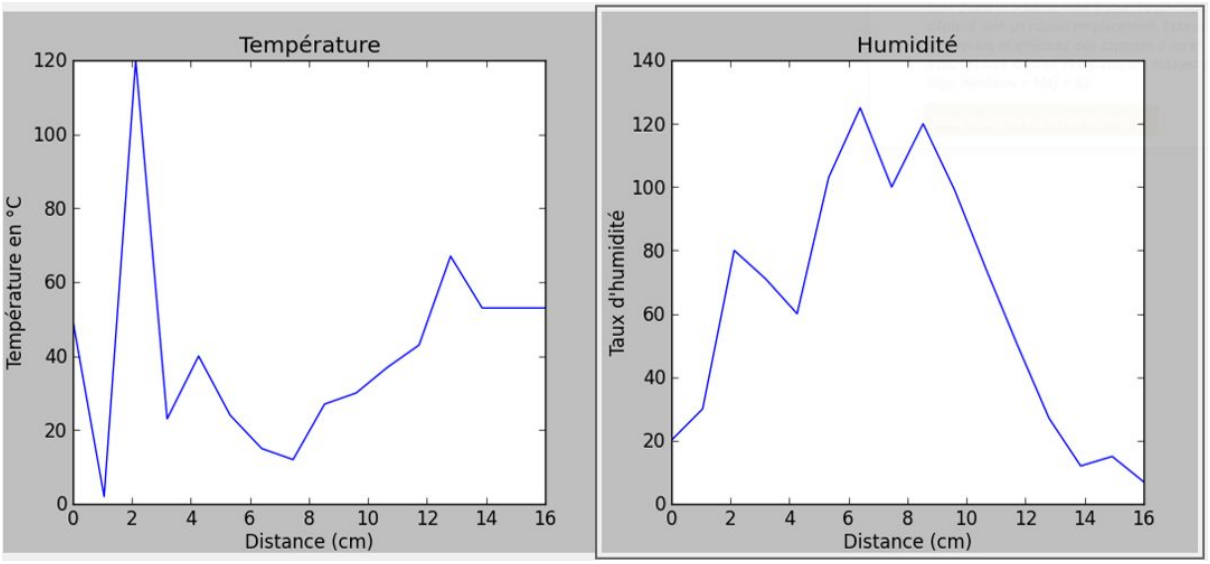


Figure : Fenêtre de commande

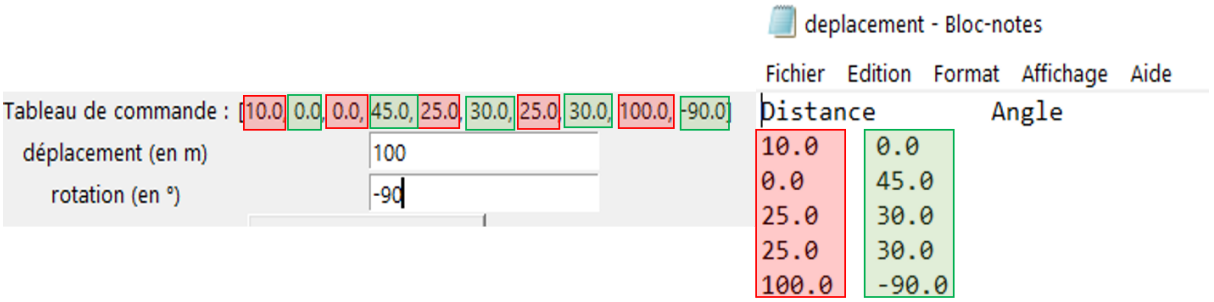
La figure ci-dessus montre la fenêtre de commande avec laquelle l'utilisateur va interagir, en (2) l'utilisateur va indiquer la distance qu'il veut que le robot avance, en (3) l'angle de rotation. Ensuite, pour ajouter cette instruction à la liste il devra appuyer sur le bouton (4). Pour créer une suite d'instruction l'utilisateur aura juste à répéter cette action plusieurs fois. Cette valeur va ensuite s'afficher en (1) de cette manière : [Distance 1, Angle 1, Distance 2, Angle 2,...]. Si l'utilisateur souhaite annuler la suite d'instruction suite à une erreur, il a juste a appuyer sur le bouton (5). Une fois que l'utilisateur est satisfait avec ces instructions il peut appuyer sur le bouton (6) pour envoyer la commande au robot (Les informations seront écrites dans un fichier .txt qui sera ensuite envoyé au robot sur Mars) .

De plus, si l'utilisateur souhaite afficher les mesures reçues par les capteurs du robot, il peut appuyer sur le bouton (7) et les deux graphique seront mis à jour avec les valeurs reçues du robot. S'affichent, à gauche, la température et à droite le taux d'humidité. (A cause du Covid-19 les valeurs présentées sont des valeurs écrites de façon aléatoire dans un fichier texte lu par le programme)



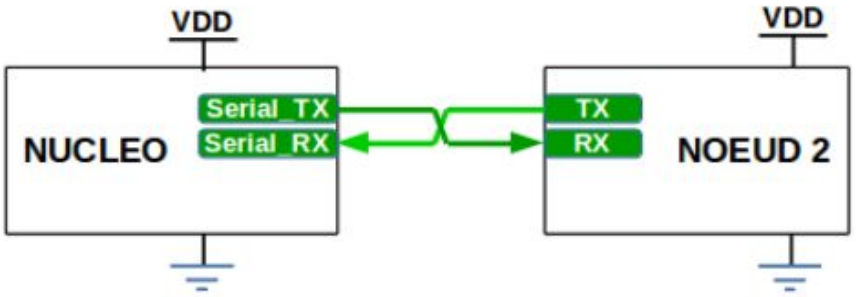
Exemple :

Sur l'image ci dessous il y a un exemple d'une suite d'instruction qui est envoyé par l'utilisateur, et comment elle est écrite dans le fichier texte.



4. Liaison PC ↔ Robot

En réalité, la liaison entre la carte Nucleo sur Terre et le robot sur Mars sera assurée par transmission radio. Pour notre prototypage à distance, on modélise la transmission radio par un protocole série de type RS-232. La transmission des données se fait sous forme de chaînes de caractères.
Dans le sens PC → Robot, la liaison permet l'envoi des commandes en angles et positions. Dans le sens opposé, la liaison permet de récupérer les mesures en température et en humidité, ainsi que les éventuels messages d'erreur. Les échanges se font en Full-Duplex, ce qui signifie qu'ils peuvent être simultanés, grâce aux deux signaux distincts RX (réception) et TX (transmission).

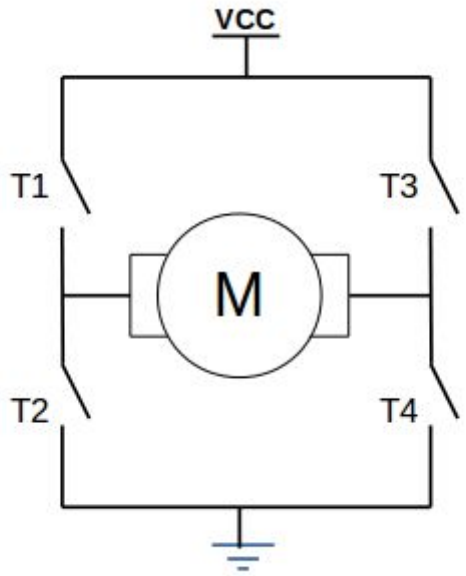


Là encore, l'enregistrement des données est réalisée dans des tableaux. Les principales difficultés que nous avons rencontrées furent :

- l'interface sur la carte Nucleo "PC" : pour faire le lien entre l'interface graphique écrite en Python et le reste du code de traitement, d'envoi/réception et de commande rédigé en langage C. Finalement, nous sommes passés par un fichier texte pour réaliser ce transfert.
- la gestion des messages d'erreur, qui ne sont pas possibles en l'état car elle nécessiterait des commandes séparées.

5. Lecture des instructions et exécution

Pour le contrôle des roues, le programme active les 2 roues (droite et gauche) séparément. En recevant les informations, celles-ci sont stockées dans un tableau sous la forme [distance, angle, distance, angle, etc…]. A chaque incrémentation du ticker, celui-ci lit une case du tableau. Si l'indice est pair, cela correspond à une distance. Alors, il va activer les 2 moteurs en même temps. Les moteurs sont chacun relié à un pont en H suivant le schéma suivant :



Les 4 transistors permettent de choisir le sens de rotation du moteur. Afin de déplacer le robot de la bonne distance, il faut calculer le temps pendant lequel on doit les activer. Tout d'abord, il nous faut choisir un rapport cyclique (qui détermine la puissance moyenne délivrée au moteur). Pour cela, il nous faudrait réaliser des tests en laboratoire, ce qui s'est évidemment avéré impossible.
Une fois ce rapport cyclique déterminé, la roue va tourner à une certaine vitesse (que l'on suppose constante dans un premier temps). En déterminant cette vitesse, il est possible de déterminer le temps pendant lequel le moteur doit être activé afin de faire avancer le robot de la distance voulue. Pour cela, on utilise la relation suivante : $temps = \frac{distance}{rayon_{roue} * vitesse_{rotation}}$.
Après ce temps écoulé, on remet le rapport cyclique à zéro, afin d'arrêter le robot.
Si la distance est négative, on fait tourner les roues dans le deuxième sens.

Si l'indice du tableau est impair, cela correspond à un angle. On n'active alors qu'un seul moteur (le droit si on veut tourner à gauche et inversement). Pour déterminer le temps pendant lequel on doit faire tourner le moteur, il faut $distance = \frac{-angle * PI}{180} * ecart_{roue}$ où $ecart_{roue}$ est l'écart entre les deux roues (en m) et on calcule le temps nécessaire de la même manière que précédemment.

Nous n'avons pas réalisé d'asservissement, donc il est probable que le robot ne soit pas extrêmement précis. Notamment, si le terrain est rocailleux ou s'il est en pente, la distance sera sûrement erronée.

Une fois qu'on a fini de parcourir le tableau, on le vide, afin que les instructions ne soient pas lues une deuxième fois.

Pour remplir ce tableau, les instructions sont reçues par liaison RS-232 (par interruption) par paquet de 20 (10 commandes de distance et 10 commandes d'angle).

6. Choix des moteurs

Afin de choisir correctement les moteurs, il nous faut calculer le couple nécessaire pour faire tourner les roues et faire avancer le robot. On fera l'hypothèse qu'un seul moteur doit être capable de faire avancer tout le robot (3 roues+le châssis) dans le cas où l'utilisateur entre une commande d'angle.

	Mars	Lense
Diamètre des roues (en m)	0,5	0,1
largeur des roues (en m)	0,2	0,03
Vitesse de déplacement des roues (m/s)	0,2	0,2
Vitesse de rotation des roues (en rad/s)	0,8	4
masse d'une roue (kg)	4,5	1,5
g	3,711	9,81
masse du rover (kg)	185	6,5
Puissance (Nm/s)	137,307	12,753
Couple (Nm)	171,63375	3,18825

Ce tableau récapitule les différentes étapes du calcul dans deux environnements : au LEnsE (c'est-à-dire sur Terre) et sur Mars. Nous avons considéré qu'il fallait des roues plus grandes sur Mars afin de pouvoir avancer plus facilement en environnement rocailleux, ce qui augmente leur poids et donc augmente le couple nécessaire.

On constate que le couple nécessaire pour faire avancer le robot sur Mars est monstrueux (environ 171 Nm). Nous n'avons pas trouvé de moteur suffisamment puissant sur le catalogue de Radiospare. Nous avons donc cherché un moteur suffisant pour tester le robot en laboratoire.

Motoréducteur c.c DOGA DO 111.9031.2B.00 / 3050 12 V 6 A 3 Nm 70 tr/min Diamètre de l'arbre: 9 mm 1 pc(s)

Référence article: 198425

Référence fabricant: DO 111.9031.2B.00 / 3050

EAN: 4016138369557

Le moteur à courant continu Doga a été développé pour des applications industrielles et s'appuie sur plus de 50 ans d'expérience et de multiples domaines d'application. La technique à aimant permanent permet une structure compacte avec de très bonnes...

Descriptif complet

Garantie: 24 mois

Fabricant: DOGA

Nous avons trouvé cette référence, qui semble convenir, car ce moteur est capable de générer un couple de 3 Nm (il sera peut-être légèrement insuffisant, auquel cas il faudrait un moteur un peu plus puissant). Il nous faudrait faire des tests en condition réelle afin de vérifier son bon fonctionnement

7. Relever la température et l'humidité

7.1. Capteur de température

Choix du capteur :

Notre robot Véronica doit nous renvoyer la température martienne (entre -128°C et +23°C). Il faut donc inclure à ce robot un capteur de température. On utilise pour cela un thermistor. Le modèle retenu est un capteur en platine P0K1.202.3FW.A.007. Il est spécifié que ce capteur varie linéairement pour des températures variant de -200°C à 300°C. Les températures martiennes sont bien incluses dans cet intervalle. La loi suivie par la résistance en fonction de la température est : $R = 3850.10^{-6}T + 100$. R est en Ω, T en kelvin.

Pour le reste de l'étude nous nous placerons entre -150°C et 50°C.
R(Tmin)=100.47 Ω
R(Tmax)= 101.24 Ω

Montage adapté :

Le capteur est calibré pour un courant $I = 1\ mA$. Pour appliquer la loi linéaire, il faut donc que le courant traversant notre composant soit très proche de $1\ mA$. La carte nucléo permet de délivrer une tension qui varie entre 0 et 3.3 V. Nous choisirons de délivrer une tension fixe de 1 V. Puisque $U = RI$, il faut que la résistance en sortie de la nucléo fasse $1k\Omega$. Nous mettons donc en série une résistance fixe de 900Ω et notre capteur, relié à la masse.

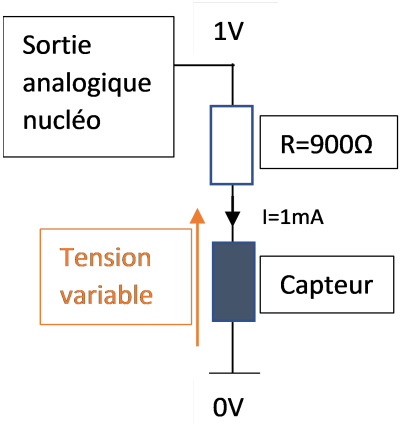


figure : Schéma électrique

La tension variable aux bornes du capteurs varie très peu :
à Tmin, Vmin=100,47 mV
à Tmax, Vmax=101.24 mV

La différence de ces deux niveaux est de $0.77\ mV$. Or, la carte nucléo est codée sur 12 bits en DAC. Il y a donc 2^{12} niveaux accessibles de tensions. Puisque la valeur maximale atteinte vaut 3.3 V, la carte nucléo ne mesure que des tensions par pallier de 0.805 mV ($= \frac{3.3}{2^{12}}$).

Ainsi l'écart entre les tensions extrémales mesurées n'est pas suffisant ! Il faut amplifier cette différence de tension.

Pour amplifier une différence de tension, nous utilisons un amplificateur différentiel. Nous choisirons le modèle simple de l'INA 122. Vous pourrez trouver sa documentation via le lien 1 : http://www.ti.com/product/INA122

Ce composant prend deux tensions en entrée : V_{in}^{+} et V_{in}^{-}. On considère que son impédance d'entrée est infinie. En sortie, il délivre une tension $V_0 = (V_{in}^{+} - V_{in}^{-}) \times G$,avec $G = 5 + \frac{200\,K}{R_g}$ un gain que l'on détermine en fixant Rg.

Déterminer Rg

Le but est que les tensions relevées couvrent la plage des tensions recevables par la carte nucléo, donc varie de 0 à 3.3 V. Nous souhaitons que la tension V_0 en sortie soit nulle pour une tension mesurée aux bornes du capteur T_{min} et que V_0 valle $3.3V$ pour une tensions mesurée T_{max} .

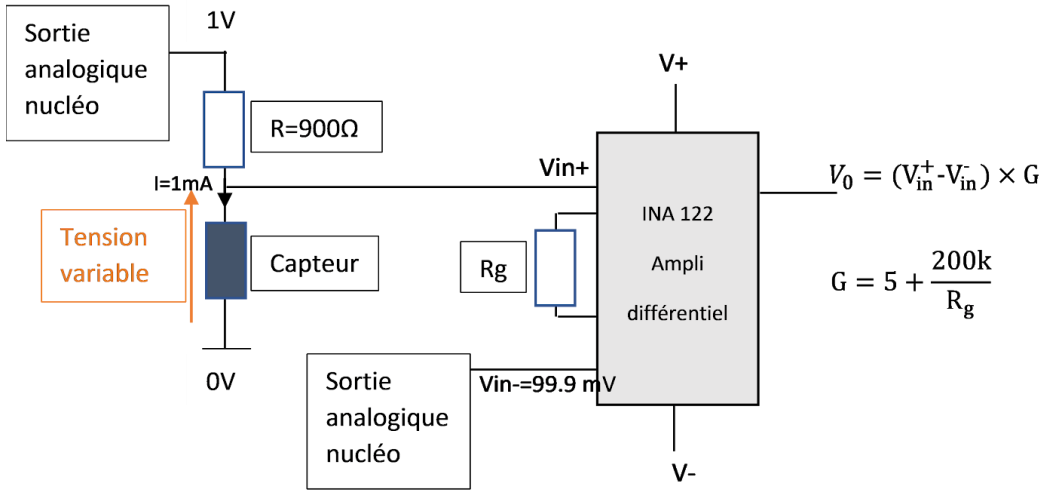
Pour répondre à ces critères, nous avons choisi de relier l'entrée V_{in}^{+} à la tension aux bornes du capteur de température martienne. Lorsque cette tension sera minimale, $V_0 = 0$, donc $V_{in}^{+} = V_{in}^{-} = 100.47\,mV$. Cette tension minimale ne peut être délivrée par la carte nucléo. La valeur la plus proche que la carte nucléo puisse délivrer est de $99.9\,mV$.

Ainsi, on obtient la valeur de $V_{in}^{-} = 99.9\,mV$ à délivrer dans l'amplificateur. Nous voulons également que lorsque $V_{in}^{+} = V_{max}$, $V_0 = 3.3V$.

On obtient : $R_g = 200k \times \frac{1}{3.3(V_{in}^{+} - V_{in}^{-}) - 5} = 81.4\ \Omega$

Nous effectuons le montage suivant :

Figure : Schéma électrique final



Ainsi, nous retrouvons la valeur de la température sur Mars avec $T = [(\frac{V_0}{G} + V_{in}^{-}) \times 10^3 - 100] \times 10^6 \times \frac{1}{3850}$, avec T en Kelvin et les tensions en Volt .

7.2. Capteur d'humidité

Il existe trois types de capteurs d'humidité (hygromètre) : les capteurs capacitifs, résistifs et thermiques.
Nous avons choisi la technologie capacitive, qui est la plus couramment utilisée et permet de mesurer l'hygrométrie de 0 à 100%. Les principaux critères de fonctionnement étant une plage de fonctionnement en température compatible avec le climat de Mars, une tension de fonctionnement adaptée à la carte Nucleo (entre 2,7 et 5,5 V) et une robustesse suffisante (résiste à la condensation, à la compensation thermique). Le capteur finalement sélectionné (IST AG HYT 271) est numérique, ce qui nous épargne de devoir réaliser un autre montage complexe d'amplification comme dans la partie précédente. Cependant, le code est plus complexe pour cette interface, qui est de type I2C. La principale difficulté rencontrée était, en l'absence de tests en salle de TP d'électronique, d'identifier les bonnes adresses à utiliser dans la documentation.

D'un point de vue code, le relevé de Température et d'Humidité se fait simultanément, à intervalle régulier, à travers l'utilisation de fonctions d'interruptions.
Le capteur d'humidité renvoie directement le taux d'humidité à chaque appel de la fonction `char humidite_getReg`, dont les dix dernières valeurs sont stockées dans le tableau `Humi`.

8. Respect du cahier des charges

Vis-à-vis du cahier des charges, certaines contraintes sont respectées et certaines ne le sont pas.

Le robot est capable d'avancer en ligne droite et de tourner angulairement.

Il stocke les données à intervalle régulier de temps (toutes les 30s). Cela ne respecte pas le cahier des charges car il aurait fallu que le robot fasse un relevé de température et d'humidité tous les 10cm. Nous n'avons pas eu le temps de faire cette conversion temps/distance parcourue. La principale difficulté vient ici du fait que le robot n'avance pas à vitesse régulière. Une simple conversion du type distance = vitesse/temps serait erronée. Il faudrait donc calculer en temps direct la distance parcourue par le robot, ce qui pourrait être fait avec un système d'asservissement (que nous n'avons pas réalisé, voir ci-dessous).

La rapidité du robot sera fixée par le rapport cyclique (et par la taille des roues). Nous n'avons pas pu faire de tests pour vérifier si les composants que nous avions choisi permettaient de respecter la valeur de 10 cm/s à 30 cm/s. Le moteur peut tourner au maximum à 70 tours/min (voir fiche technique). On peut donc faire tourner le moteur à plus d'un tour/s. Si la roue a un périmètre supérieur à 10 cm (c'est à dire un diamètre d'environ 3.2 cm, la contrainte de vitesse est respectée). Cependant, ces calculs sont valables pour un sol plat et lisse. Comme expliqué ci-dessous, il aurait fallu mettre en place un asservissement pour s'assurer du respect de la contrainte en toute situation.

Le robot n'est pas asservi. Ainsi, il sera très sensible aux variations de terrain et aux variations de pentes, qui auront tendance à ralentir le robot. Il devient alors difficile de s'assurer que la distance parcourue est égale à la commande entrée par l'utilisateur.

De même, nous n'avons pu installer de batterie sur le robot donc nous n'avons pas travaillé sur son autonomie.

En ce qui concerne les capteurs : le capteur de température fonctionne de -200°C à +300°C, ce qui est compatible avec les températures présentes à la surface de Mars (de -128° à +23°C). De même, le capteur d'humidité permet de relever des températures entre 0% et 100%, ce qui est pratique car nous n'avons même pas besoin de vérifier les plages d'humidité habituellement trouvables sur Mars pour s'assurer du fonctionnement du capteur.

L'interface graphique est utilisable sans formation préalable et assez intuitive. Elle permet à la fois d'envoyer les instructions au robot et d'afficher en temps réel (Sous demande de l'utilisateur) les relevés de température et d'humidité.

9. Compétences acquises

DREVET MULARD Marie

Pour le capteur de température, j'ai appris à bien lire la documentation technique des composants. J'ai ainsi appris les différents fonctionnement de capteur de température et d'humidité.

Je me suis renseignée sur les caractéristiques précises de la nucléo grâce aux informations du Lense.

http://lense.institutoptique.fr/nucleo-tester-ma-premiere-application/

Ensuite, Eitan et moi nous sommes intéressés au contrôle angulaire et rectiligne des roues du robot, via le moteur. Pour cela nous nous sommes appuyés sur le tutoriel du Lense :

http://lense.institutoptique.fr/nucleo-tester-ma-premiere-application/

GUITTON Adrien

Pour les interfaces graphiques j'ai suivi les même tutoriels que Eitan ainsi que d'anciens TDs que j'ai eu en classe préparatoire pour comprendre les bases. Pour ce qui est de l'affichage des graphiques avec Tkinter j'ai suivi beaucoup de tutoriels différents mais j'ai trouvé les plus utiles:
https://datatofish.com/matplotlib-charts-tkinter-gui/
https://matplotlib.org/tutorials/introductory/pyplot.html
https://matplotlib.org/tutorials/introductory/pyplot.html
Ensuite comme méthode pour lier le C et le Python on a choisi de passer par des fichiers textes. J'ai suivi des tutoriels pour savoir lire et écrire des fichiers textes avec les deux langages de programmation.
Pour Python : http://www.xavierdupre.fr/app/teachpyx/helpsphinx/c_io/files.html
Pour le C :
http://lense.institutoptique.fr/langage-c-td-s5-2/#Seance_5_Lire_des_fichiers_ASCII
Puis, comme j'ai aussi codé le code pour la lecture du fichier texte en C, et comme on a simulé la liaison entre le robot et le PC par une liaison RS232, je me suis donc renseigné avec le tutoriel suivant :
http://lense.institutoptique.fr/nucleo-configurer-une-communication-point-a-point-de-type-rs232-2/

MOULONGUET Aymeric

Marie et moi avons suivi des tutoriels d'utilisation du logiciel de simulation LTSpice afin de simuler certaines composantes de notre circuit électronique analogique. Bien que les tutoriels soient suffisamment détaillés, nous n'avons pas réussi à aller au-delà de la simple reproduction sur nos PC des exemples donnés. En particulier, nous n'avons pas réussi à trouver les composants (capteurs) que nous souhaitions utiliser ou leur équivalents dans les bibliothèques de LTSpice. Nous nous sommes donc familiarisés avec ce logiciel mais n'avons pu ensuite l'utiliser efficacement pour notre projet.
Pour le capteur numérique d'humidité, je me suis servi des tutos du LEnsE pour effectuer la communication avec la carte Nucleo en I2C : http://lense.institutoptique.fr/nucleo-configurer-un-reseau-adressable-de-type-i2c-3/
Pour ce qui est de la liaison RS232 entre les 2 cartes Nucleo, Adrien et moi nous sommes aidés du tuto correspondant :
http://lense.institutoptique.fr/nucleo-configurer-une-communication-point-a-point-de-type-rs232-2/

PECHEVIS Eitan

J'ai suivi des tutos pour développer une interface graphique sous Python.

Parmi eux, https://python.doctor/page-tkinter-interface-graphique-python-tutoriel . Cela m'a permis d'apprendre les bases, notamment comment organiser correctement une fenêtre graphique. Pour ce qui concerne les graphiques, j'ai trouvé une façon de les afficher en utilisant le module matplotlib, qui est très efficace. Cependant, c'est Adrien qui s'est chargé de les intégrer à la fenêtre Tkinter.
J'ai également suivi le tuto du LEnsE "Faire varier la vitesse d'un moteur à courant continu", afin de pouvoir contrôler les roues du robot.
http://lense.institutoptique.fr/nucleo-faire-varier-la-vitesse-dun-moteur-a-courant-continu-3/
Marie et moi avons ensuite écrit les programmes mbed permettant le contrôle des roues en fonction des commandes entrées par l'utilisateur.

10. Retour d'expérience

Ce projet s'est déroulé entièrement en télétravail. Nous avons pu réaliser l'apprentissage théorique : le choix des composants, la lecture des tutoriels, le codage.

Nous avons réussi à organiser des plages horaires pour travailler tous les 4 ensemble. Le travail en équipe s'est très bien passé. Nous déplorons les quelques quiproquos ou temps d'explication plus longs que d'habitude dû au manque de visibilité sur le travail des autres. Cependant, de façon générale, la communication au sein de l'équipe a été efficace. Nous avons pris l'habitude, à chaque séance, de faire le point sur l'avancement des "sous-missions" en cours et sur celles qu'il restait à accomplir. Ensuite, on se répartissait le travail en binôme pour gagner en efficacité, avec un point d'équipe en milieu et fin de séance.

En revanche, nous n'avons pas pu acquérir toutes les connaissances pratiques prévues initialement. N'ayant pas accès au laboratoire, il nous était impossible d'observer les bons comportements de nos composants, ni de tester le fonctionnement des programmes au-delà des erreurs de compilation. Ce fut très limitant et handicapant. En effet, nous pensons que nos programmes auraient été plus performants en conditions normales, car la détection des problèmes et les tests des solutions auraient été plus rapides. Bien sûr, nous savons que ces problèmes de distanciation sont regrettables et ne pouvaient être résolus. En conséquence, nous étions parfois découragés de travailler sans observer les améliorations. Nous savions que le but final - implémenter un prototype réel - ne pourrait être atteint.

Nous pensons que ce mode de fonctionnement a également affecté le dialogue que nous avions avec les professeur.e.s. Bien qu'ils se soient mobilisés au maximum

pour être présents, les délais entre chaque intervention étaient assez longs et il n'était pas évident d'avoir un retour si nous n'avions pas de questions précises.

11. Annexes

Annexe - CODES

```
#include "mbed.h"
```

Bibliothèque spécifique pour capteurs

```
#define  capt_hum     0x28                    // Capteur d'humidité I2C
```

Bibliothèque spécifique pour Contrôle des roues

```
#include "math.h"
```

Entrées et Sorties pour capteurs

```
/* Pour le capteur de température */
AnalogOut sortie_3V(PA_4);      //connecté au pont diviseur de tension
AnalogOut sortie_100mV(PA_5);  //connecté à l'entrée de l'ampli différentiel

AnalogIn entree_analog(A0);          //Relevé tension capteur température
AnalogOut sortie_analog(D13);        //inutilisé actuellement
I2C my_i2c(D14, D15);          //Relevé valeur capteur température
Serial pc(SERIAL_TX, SERIAL_RX);
```

Entrées et sorties pour Contrôle des roues

```
PwmOut roueG_sens1(D10);       //Pour ces 4 lignes, il faut déterminer les
broches qui sont reliées aux deux ponts en H
PwmOut roueG_sens2(D11);  //pouvoir faire tourner les roues dans les 2 sens
PwmOut roueD_sens1(D8);
PwmOut roueD_sens2(D9);
AnalogIn signal(A1);           //Récupération du tableau de commande
```

Déclaration des variables pour les capteurs

```
double tens4=3;          //V tension analogique associée à sortie_3V
double tens5=0.10028;    //V tension analogique associée à sortie_100mV

int meas_int;
float value;
float Temp[20];          //Tableau de 20 valeurs de température pour 10 min
float Humi[20];          //Tableau de 20 valeurs d'humidité pour 10 min
int j;
int k1, k2;
```

Déclaration des variables pour le Contrôle des roues

```
double meas;         //Définition des variables utiles pour le programme
float angle;
float distance;
float ecart_roue = 0.5;    //Valeur de la distance entre les roues (en m)
float rayon_roue = 0.2;        //Valeur du rayon d'une roue (en m).
```

```
float rotation_roue = 1;  //Vitesse de rotation des roues (en m). En réalité, celle-ci est reliée    au rapport cyclique du moteur.
float rapport_cyclique;    //Pour déterminer le rapport cyclique nécessaire, il faudrait faire
                                //des tests directement avec les moteurs.
```

Déclaration des variables pour la liaison mixte

```
float Temp[20];          //Tableau de 20 valeurs de température pour 10 min
float Humi[20];          //Tableau de 20 valeurs d'humidité pour 10 min
float Dist[10];          //Instructions en distance
float Angl[10];          //Instructions en angle
float Mix[20];
int i;
int k;
int test;
```

Déclaration du ticker pour les capteurs

```
Ticker echant;                  //Déclaration du ticker
```

Déclaration des fonctions associées aux capteurs

```
void Stock20(void);   //Déclaration de la fonction d'interruption du ticker
char device_reg;
```

Déclaration du ticker pour le Contrôle des roues

```
Ticker roue;
```

Déclaration des fonctions pour le Contrôle des roues

```
void roue(void);
```

Déclaration du ticker Liaison mixte

```
Ticker emission;   // Déclaration du ticker
```

Déclaration des fonctions Liaison mixte

```
void envoi(void);  //Déclaration de la fonction d'interruption du ticker
void recoi(void);


int main(){
```

Bloc main associé aux capteurs

```
    sortie_3V.write(tens4);
    sortie_100mV.write(tens5);
    for (j=0 ; j< 20 ; j++){          //Initialisation des valeurs à 0
            Temp[j] = 0;
            Humi[j] = 0; }
    echant.attach(&Stock20, 30);
```

Bloc main associé au Contrôle des roues

```
 meas=signal.read();
  roueG_sens1.period_ms(0);    //On initialise les vitesses des moteurs à 0 pour être
  roueD_sens1.period_ms(0);    //sûr d'être à l'arrêt au lancement du programme.
  roueG_sens1.write(0);
  roueD_sens1.write(0);
```

```
  roueG_sens2.period_ms(0);
  roueD_sens2.period_ms(0);
  roueG_sens2.write(0);
  roueD_sens2.write(0);
  l = 0;                        //Indice permettant de lire le tableau
  n = length(signal);
 roue.attach(&commande, 0.01); //Appel du ticker et définition
```

Bloc main associé à la liaison mixte

```
  pc.baud(115200);
  test = 0;
    for (i=0 ; i< 10 ; i++){          //Initialisation des valeurs à 0
            Dist[i] = 0;
            Angl[i] = 0;}
    for (i=0 ; i< 20 ; i++){          //Initialisation des valeurs à 0
            Mix[i] = 0; }
    emission.attach(&envoi, 600);     //envoi des données T et H toutes les 10
min
   pc.attach(&recoi);           //réception des commandes Angles et Distances
(réception //par interruption)

        while(1) { }
}
```

Fonction des capteurs

```
char humidite_getReg(char ad_reg){          //Recevoir une mesure d'humidité
    char data_reg;
    device_reg = ad_reg;
    k1 = my_i2c.write(capt_hum << 1, &device_reg, 1, true);    //lecture de
l'adresse
    k2 = my_i2c.read(capt_hum << 1, &data_reg, 1, false);    //écriture de la
donnée
    return data_reg;
}


void Stock20() {                        // Fonction d'interruption du ticker
char humi_reg = 0;
humi_reg = humidite_getReg(capt_hum);
meas_int = entree_analog.read();     // sur 12 bits MSB

for (j=0; j<19; j++){
    Temp[j]=Temp[j+1];
    Humi[j]=Humi[j+1];
    }
    Temp[19]=((meas_int/2462+0.0999)*1000-100)*260; //loi linéaire :
Température [en °C] = f(Tension [en V])
    Humi[19]=humi_reg;               //relève l'humidité
    }
```

Fonction associée au Contrôle des roues

```
void commande(){
if (test==0){

  global signal;
  if (l >= n){
      test=1;
      pc.printf("En attente de nouvelles instructions.");
      l = 0;                                          //On            réinitialise
l'indiçage lorsqu'on est arrivé au bout du //tableau
      signal = [];                                      //On vide le tableau
afin de ne pas effectuer les instructions //une deuxième fois.
      signal
  }
  else{
      if (l % 2){                                        //On   regarde   si
c'est un angle ou une distance
      angle = signal[l];                               //Lecture    de    la
valeur de l'angle
      if (angle<0){
      distance = -angle*M_PI/180*ecart_roue;           //Calcul     de     la
distance à parcourir avec la //roue droite
      roueG_sens1.period_ms(0);
      roueD_sens1.period_ms(10);
      roueG_sens1.write(0);
      roueD_sens1.write(rapport_cyclique);
      delay(1000*distance/(rayon_roue*rotation_roue));  //On  suppose  que  le
delay est en ms (à //vérifier)
      routeD_sens1.write(0);
      l++;
      }
      else{
      distance  =  angle*M_PI/180*ecart_roue;           //Calcul  distance  à
parcourir avec la roue gauche
      roueG_sens1.period_ms(10);
      roueD_sens1.period_ms(0);
      roueG_sens1.write(rapport_cyclique);
      roueD_sens1.write(0);
      delay(1000*distance/(rayon_roue*rotation_roue)); //delay est en ms
      roueG_sens1.write(0);
      l++;
      }
      }
      else{
      distance = signal[l];    //Lecture la valeur de distance à parcourir
      if (distance < 0){
      roueD_sens1.period_ms(0);
      roueG_sens1.period_ms(0);
      roueG_sens1.write(0);
      roueD_sens1.write(0);
      roueD_sens2.period_ms(10);
```

```
      roueG_sens2.period_ms(10);
      roueG_sens2.write(rapport_cyclique);
      roueD_sens2.write(rapport_cyclique);
      delay(-1000*distance/(rayon_roue*rotation_roue)); //On suppose que le
delay est en ms
      roueD_sens2.write(0);
      roueG_sens2.write(0);
      l++;
      }
      else{
      roueD_sens2.period_ms(0);
      roueG_sens2.period_ms(0);
      roueG_sens2.write(0);
      roueD_sens2.write(0);
      roueD_sens1.period_ms(10);
      roueG_sens1.period_ms(10);
      roueG_sens1.write(rapport_cyclique);
      roueD_sens1.write(rapport_cyclique);
      delay(1000*distance/(rayon_roue*rotation_roue)); //On suppose que le
delay est en ms
      roueD_sens1.write(0);
      roueG_sens1.write(0);
      l++;
      }
      }
  }
}
```

Fonctions associées à la liaison mixte

```
void envoi() {                          // Fonction d'interruption du ticker
    for (i=0; i<19; i++){
        pc.printf("%lf\t%lf\r\n", Temp[i], Humi[i]); }
    }

void recoi() {
if(test==1){
for (k=0; k<10;k++){                  //On décale les valeurs pour obtenir un
nouveau tableau
    //sscanf(pc.gets(),"%lf\t%lf\r\n", &Dist[9], &Angl[9]);
    pc.scanf("%lf\t%lf\r\n", &Dist[9], &Angl[9]);
    for (i=0; i<9; i++){
        Dist[i]=Dist[i+1];
        Angl[i]=Angl[i+1];}
}
    for (i=0; i<20; i++){
            if(i%2){
            mix[i] = Angl[i//2+1];
            }
      else{
```

```
        mix[i] = Dist[i//2]
        }
        }
        test=0
}
```

Programme IHM (Interface Homme Machine) sous Tkinter par Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 23 14:00:00 2020

@author: Eitan_Pechevis & Adrien_Guitton
"""
```

Bloc importation des bibliothéque

```
import numpy as np
import matplotlib.pyplot as plt \\ Module pour tracer des courbes
import tkinter \\ Module pour la création d'interface graphique
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg \\
Module pour lié tkinter et graphique
from matplotlib.figure import Figure \\ Module pour paramétrer la
taille des fenètre graphique
```

Bloc déclaration des variables

```
A=[] \\ Liste qui va contenir valeurs rentrées par l'utilisateur
Instruction = [] \\ Liste qui va contenir les valeurs de position
et d'angle qui va être envoyé au robot
infos=[] \\ Liste contenant les informations lu dans le fichier
texte contenant les information du capteur
temp=[] \\ Liste contenant les informations de température
humi=[] \\ Liste contenant les informations de taux d'humidité
```

Bloc création de l'interface graphique

```
fen1 = tkinter.Tk() \\ Création de la fenêtre principal de
l'interface graphique
fen1.geometry("1120x800") \\ Définition en pixel de la taille de
la fenétre

figure1 = Figure(figsize=(7, 5), dpi=100) \\ Création de la
fenêtre du graphique n°1 (Température)
figure2 = Figure(figsize=(7, 5), dpi=100) \\ Création de la
fenêtre du graphique n°2 (Taux d'humudité)


axes1 = figure1.add_subplot(111) \\ Création des données pour la
graphique n°1 (Température)
```

```
axes2 = figure2.add_subplot(111) \\ Création des données pour la graphique n°2 (Taux d'humudité)

axes1.set_title("Température") \\ Création du titre pour le graphique 1
axes1.set_xlabel("Distance (cm)") \\ Création de l'axe x pour le graphique 1
axes1.set_ylabel("Température en °C") \\ Création de l'axe y pour le graphique 1

axes2.set_xlabel("Distance (cm)") \\ Création du titre pour le graphique 2
axes2.set_ylabel("Taux d'humidité") \\ Création du titre pour le graphique 2
axes2.set_title("Humidité") \\ Création du titre pour le graphique 2

canvas1 = FigureCanvasTkAgg(figure1, fen1) \\ Implémentation du de la fenêtre graphique n°1 avec Tkinter
canvas1.draw() \\ Afficher le graphique n°1 au démarrage
canvas1.get_tk_widget().place(x=10, y =220, width=550, height=500)
\\ Définition de la taille et de la position du graphique n°1
\\ Même chose pour graphique n°2
canvas2 = FigureCanvasTkAgg(figure2, fen1)
canvas2.draw()
canvas2.get_tk_widget().place(x=550, y =220, width=550, height=500)

\\ Création des texte d'instruction
titre = tkinter.Label(text = "Panneau de commande", fg = "black", relief = "raised")
texteTableau = tkinter.Label(text = "Tableau de commande :")
affichageA = tkinter.Label(text=str(A))
affichagePos = tkinter.Label(text="déplacement (en m)")
affichageRot = tkinter.Label(text="rotation (en °)")

\\ Création des bloc de texte où l'utilisateur va marquer les instructions
positionValeur = tkinter.Entry()
rotationValeur = tkinter.Entry()

\\ Création des différents boutons de l'interface
boutonAjouter = tkinter.Button(fen1,text = "Ajouter cette instruction")
boutonAnnuler = tkinter.Button(fen1,text = "Tout annuler")
boutonEnvoyer = tkinter.Button(fen1,text = "Envoyer cette suite d'instruction")
```

```
boutonGraph = tkinter.Button(fen1, text = "Affiche courbe")
```

Fonction pour récupérer les valeurs rentrée par l'utilisateur

```
def fct1():
      global A
      position = float(positionValeur.get()) \\ récupération de la
valeur de position
      rotation = float(rotationValeur.get()) \\ récupération de la
valeur de rotation
      A.append(position)
      A.append(rotation)
      affichageA.config(text=str(A)) \\ modification du label

boutonAjouter.config(command=fct1) # assignation de la fonction
fct() au bouton \\ Appel à la fonction sur le bouton "Ajouter"
```

Fonction pour remettre à 0 la liste d'instruction

```
def fct2():
      global A
      A=[]
      affichageA.config(text=A)
boutonAnnuler.config(command=fct2) \\ Appel à la fonction sur le
bouton "Annuler"
```

Fonction qui écrit les instruction de déplacement sur le fichier texte "deplacement.txt"

```
def fct3():
      global A
      global Instruction
      Instruction[:] = A[:]
      ecriture = open("deplacement.txt", "w") \\Ouverture du
fichier texte en mode écriture
      Necr=len(Instruction)//2
      for k in range (1,Necr):

ecriture.write('\n'+str(Instruction[2*k])+'\t'+str(Instruction[2*k
+1])) \\Ecriture dans le fichier texte des instruction de distance
et d'angle
            ecriture.close() \\ Fermuture du fichier texte

boutonEnvoyer.config(command=fct3) \\ Appel à la fonction sur le
bouton "Envoyer"
```

Fonction qui va lire les donnés sur le fichier texte "data.txt" et les affichées

```
\\ c1,f1,ax1 représentent le canvas, la figure et la fenêtre du
graphique du graphique 1, même chose pour c2,f2,ax2
```

```
def fct4(c1,f1,ax1,c2,f2,ax2):
      global humi
      global temp
      global infos
      ax1.cla() \\Efface le précédent graphique qui se situe de la
figure 1
      ax2.cla() \\ Même chose pour la figure 2
      lecture = open("data.txt","r") \\ Ouverture en mode lecture
du fichier "data.txt"
      texte = lecture.read() \\ Insertion du texte du fichier dans
la variable texte
      lecture.close() \\ Fermeture du fichier
      a=texte.split('\n') \\On sépara la variable texte pour créer
une liste a qui contient chaque ligne du texte
      for i in range(0,(len(a))):
            don=a[i].split("\t") \\ On sépare chaque ligne en deux
valeur (Séparé par une tabulation)
            infos.append(don[0])
            infos.append(don[1])

      Nlec=len(infos)//2

      for k in range (1,Nlec): \\ Création des deux listes humi
(Taux d'humidité) et temp (Température)
            humi.append(infos[2*k])
            temp.append(infos[2*k+1])
      Nx=len(temp)
      distance = ((Nx//2)-1)*10
      x = np.linspace(0, Nx,Nx)
      ax1.plot(x, humi)
      ax2.plot(x,temp)

      ax1.set_title("Température")
      ax1.set_xlabel("Distance (cm)")
      ax1.set_ylabel("Température en °C")

      ax2.set_xlabel("Distance (cm)")
      ax2.set_ylabel("Taux d'humidité")
      ax2.set_title("Humidité")
      \\ Affichage des valeurs sur les deux figures
      c1.draw()
      c2.draw()
      humi=[]
      temp=[]
      infos=[]
```

```
boutonGraph.config(command=lambda
:fct4(canvas1,figure1,axes1,canvas2,figure2,axes2))
\\Placement des labels (Bouttons et texte) dans la fenétre
graphique
titre.grid(column = 0, row = 0, columnspan = 2)
texteTableau.grid(column = 0, row = 1)
affichageA.grid(column = 1, row = 1)
affichagePos.grid(column = 0, row = 2)
positionValeur.grid(column = 1, row = 2)
affichageRot.grid(column = 0, row = 3)
rotationValeur.grid(column = 1, row = 3)
boutonAjouter.grid(column = 0, row = 4, columnspan = 2)
boutonAnnuler.grid(column = 0, row = 5, columnspan = 2)
boutonEnvoyer.grid(column = 0, row = 6, columnspan = 2)
boutonGraph.grid(column = 0, row = 7, columnspan =2)

\\ Fermeture de la fenêtre 1
fen1.mainloop()
```