Rapport technique
Robot Véronica

ProTIS

Groupe 2.6

Fisson Lisa
Denain Marius
Surmeli Bora
WIECZOREK Louis

Nous attestons que ce travail est original, que nous citons en référence toutes les sources utilisées et qu'il ne comporte pas de plagiat.

6 avril 2023

Table des matières

1 Introduction

L'objectif est de réaliser un prototype d'un système embarqué. Le système que l'on a choisi est le Robot Veronica. Ce document a pour but de détailler les fonctions du robot Veronica ainsi que d'expliquer les points techniques de ses fonctionnalités, de la conception à l'intégration sur le prototype.

1.1 Présentation du système

Le robot Veronica est un système faisant partie de la catégorie robot embarqué. Il s'agit d'un robot guidé à distance permettant d'explorer le sol particulier de Mars dans le but de détecter la présence d'eau ou d'autres substances.

Ce robot est basé sur un système à deux roues indépendantes et motorisées ainsi qu'une roue libre pour se déplacer. Des données d'humidité et de température relevées par le robot peuvent être transmises à la base. Son parcours, composé de tronçons de ligne droite et de virages, est transmis au fur et à mesure depuis une base terrienne. Les données collectées sont ensuite enregistrées et transmises à intervalle régulier.

1.2 Cahier des charges fonctionnel

Les contraintes et fonctionnalités que doit respecter le système sont regroupées dans le cahier des charges fonctionnel ci-dessous.

Critère	Fonction	Tolérance
Déplacement	Le robot doit être capable d'avancer en ligne droite selon une commande transmise depuis un système distant	La vitesse du robot doit être comprise entre 10 cm/s et 30 cm/s
Direction	Le robot doit être capable d'effectuer des rotations sur lui-même d'un angle transmis depuis un système distant	L'erreur maximale tolérée est de 2 cm sur la position et de 3° sur l'angle
Autonomie	Le robot doit pouvoir être autonome	Le robot doit pouvoir réaliser un parcours de 1km sans que ses batteries ne soient rechargées
Mesure	Le robot doit être capable de mesurer la température et l'humidité	Les données doivent être prises tous les 10 cm
Ergonomie	Les données doivent pouvoir être affichées en fonction de la distance ou du temps	L'interface Humain-Machine, permettant de transmettre les ordres de parcours, doit pouvoir être utilisée sans formation préalable.

TABLE 1 – Cahier des charges fonctionnel du système Robot Veronica

1.3 Schéma fonctionnel du système

Les solutions choisies afin que le système respecte toutes les fonctionnalités récapitulées dans le cahier des charges fonctionnel sont développées et illustrées sur le schéma fonctionnel suivant.

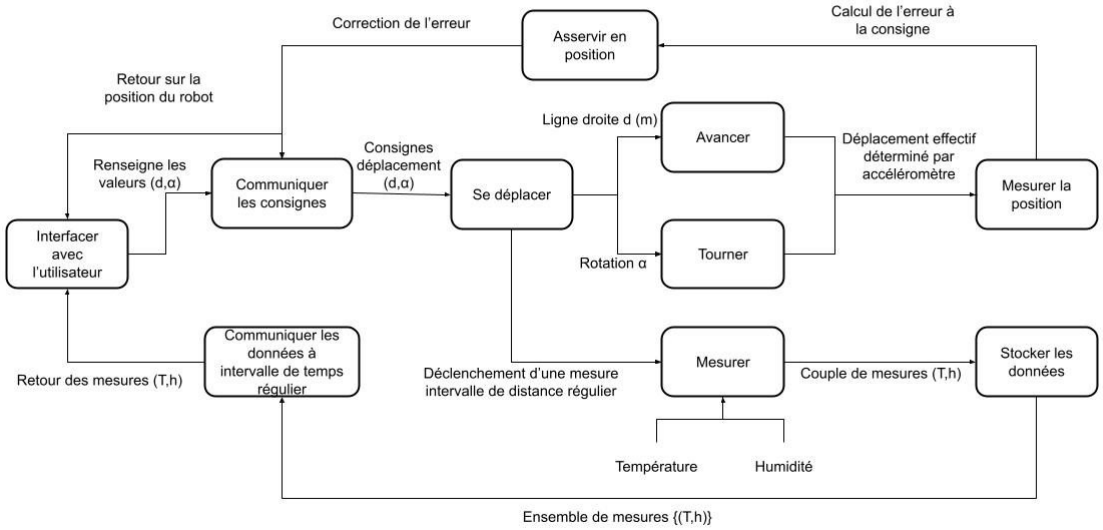


FIGURE 1 – Schéma fonctionnel du système Robot Veronica

2 Description technique du système

Cette section a pour but de d'expliquer en détail les fonctions qui ont été implémentées dans le robot Veronica.

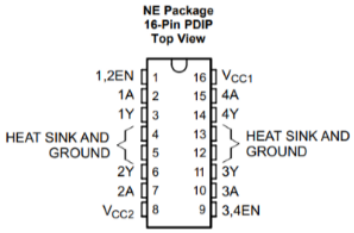
2.1 Pilotage du robot

2.1.1 Déplacement en ligne droite

Le pilotage du robot nécessite plusieurs fonctionnalités, dont la fonction "Déplacement en ligne droite". Pour cela, le robot est constitué de deux moteurs POLOLU3239 alimentés en 12V par batterie externe, deux ponts en H L293D permettant de faire avancer ou reculer les roues et une carte nucléo L476RG contenant le code en C enbarqué et permettant le déplacement du robot en autonomie. Les ponts en H sont directement intégré sur une carte pour plus de praticité. Vous trouverez en figure 2 les pin du pont en H.

Nous avons réalisé ce montage pour chacun des moteurs :

Les entrées EN (enable) permettent d'actionner les moteurs. Les deux entrée EN des deux moteurs sont reliées entre elles sur la carte. Elles sont directement reliées



Pin Functions

PIN		TYPE	DESCRIPTION
NAME	NO.		
1,2EN	1	I	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	I	Driver inputs, noninverting
<1:4>Y	3, 6, 11, 14	O	Driver outputs
3,4EN	9	I	Enable driver channels 3 and 4 (active high input)
GROUND	4, 5, 12, 13	—	Device ground and heat sink pin. Connect to printed-circuit-board ground plane with multiple solid vias
V_{CC1}	16	—	5-V supply for internal logic translation
V_{CC2}	8	—	Power VCC for drivers 4.5 V to 36 V

FIGURE 2 – Schéma pin du pont en H

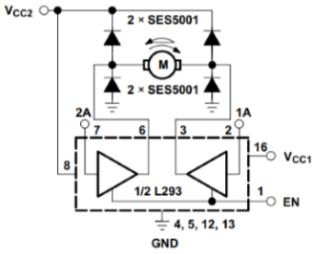


Figure 10. Bidirectional DC Motor Control

Table 3. Bidirectional DC Motor Control

EN	1A	2A	FUNCTION(1)
H	L	H	Turn right
H	H	L	Turn left

(1) L = low, H = high, X = don't care

FIGURE 3 – Schéma câblages pont en H et moteur

à la sortie PB_3 de la carte nucléo. Les entrées 1A et 2A du moteurs 3 sont reliées respectivement aux sorties PC_9 et PB_8 de la carte nucléo, et les entrées 3A et 4A du moteur 2 sont reliées aux sorties PB_13 et PB_14, toutes de type PWM. Ces entrées permettent d'une part de faire avancer ou reculer les roues, et d'autre part de gérer la vitesse en faisant varier le rapport cyclique (cf partie code). Attention à ne pas essayer de réguler de la vitesse en jouant sur le rapport cyclique le l'entrée EN car les deux moteurs, n'étant pas identiques, ne tournent pas à la même vitesse. Le rapport cyclique de l'entrée EN reste donc constant à 1. Par la suite, il est donc indispensable de réaliser un asservissement en vitesse sur chacun des moteurs.

2.1.2 Asservissement en vitesse

Un asservissement en vitesse est nécessaire afin d'amener les moteurs à tourner selon une vitesse de consigne et de permettre au robot d'avancer en ligne droite.

Les sorties A et B des encodeurs de chacun des moteurs sont récupérées par des entrées de type InterruptIn sur la carte nucléo. Une fonction compteur vient ensuite

compter le nombre de fronts montants des sorties A sur un temps défini (0,5 s). Ce nombre de fronts montants est ensuite comparé à une consigne. Une erreur sur le nombre de fronts montants $C_{consigne} - C_{sortie}$ est calculée pour chaque moteur. Un asservissement par régulateur PI est alors réalisé sur chacun des moteurs : $rc = rc + (Kp * e + Ki * ei)$. Avec rc le rapport cyclique des sortie PWM, $Kp = 0,001$ le coefficient proportionnel, e l'erreur, Ki = 0,0001 le coefficient intégral et ei l'erreur intégrale, soit la somme des erreurs.

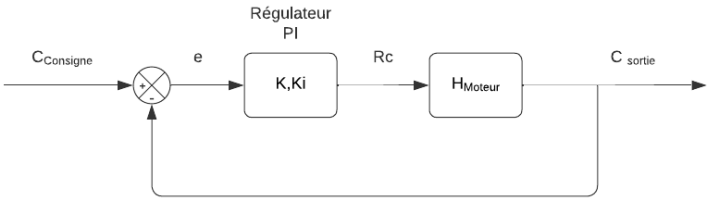


FIGURE 4 – Schéma bloc asservissement en boucle fermée

Afin de vérifier l'asservissement, il est possible de lier la carte nucléo par câble USB et d'afficher la valeur des erreurs sur les deux moteurs. Si on bloque l'une des roues, la valeur de l'erreur augmente puis on observe une accélération lorsqu'on relâche la roue et un retour vers une erreur nulle.

2.1.3 Alimentation

L'alimentation des ponts en H, moteurs et carte nucléo se fait par batterie 12V. La batterie est directement reliée à la carte support qui vient ensuite alimenter chaque composant. Attention, la carte nucléo peut être alimentée soit par alimentation externe, soit par câble USB. Il est indispensable de placer le cavalier JP5 sur la position correspondante (cf figure 5)

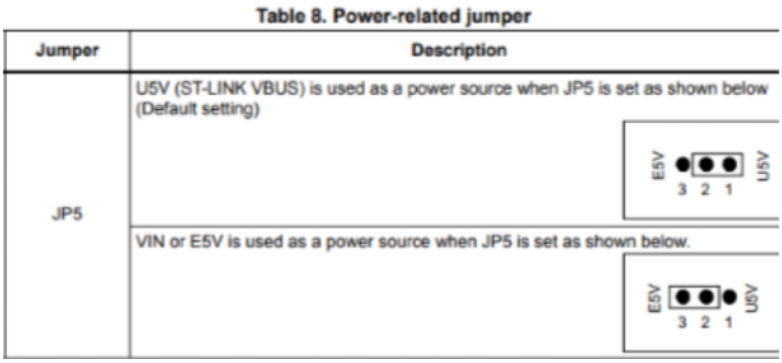


FIGURE 5 – Position cavalier JP5 en fonction du type d'alimentation

2.1.4 Code

Le code implémenté sur la carte nucléo est en C/C++. Il a été réalisé sur Mbed et a été transféré ensuite sur la carte nucléo via câble USB. Les affichages (printf) se

font sur TéraTerm. Cette partie permet d'expliquer les différentes portions du code, qui se trouve au complet en annexe.

Entrées/Sorties : Les sorties de la carte nucléo contrôlant le pont en H (entrées pour le pont en H) sont de type *PwmOut*. Elles sont définies en début de code, avant la fonction *main*.
Ex : PwmOut moteur_EN(PB_3) ; // Sortie EN
Avec *PwmOut* le type de la sortie, *moteur_EN* le nom et *PB_3* la broche associée.
Les entrées de la carte nucléo (sorties du pont en H) étant le retour des encodeurs sont de type *InterruptIn*.
Ex : InterruptIn moteur_A2(PB_2) ; // Retour de l'encodeur A du moteur 2

Variables générales : Les variables générales sont aussi définies en amont de la fonction *main*. Le rapport cyclique (rc) ainsi que les coefficients proportionnel (Kp) et intégral (Ki) sont des réels de type *double*.
Ex : double rc2=0 ; //Rapport cyclique du moteur 2, pilotage en vitesse
Les compteurs (C) permettant de compter le nombre de fronts montant et les erreurs (e) sont des entiers de type *int*.
Ex :int e2 = 0 ; //Erreur sur le moteur 2 initialisée à 0

Déclaration du ticker : Le *ticker* permet d'exécuter une fonction à intervalle de temps régulier. Il sera donc par la suite attaché à une fonction d'interruption.
Ex :Ticker toggle_V_ticker ; //Déclaration du ticker

Fonctions : En C et C++, les fonctions sont définies après la fonction *main* (ou dans un autre fichier, importé dans le main) et doivent être déclarées avant la fonction *main*.

La fonction ***Compteur*** permet de compter le nombre de front montant. Cette fonction permet simplement d'incrémenter un compteur (C). *Ex :void compteur (void) {*
C=C+1 ;
}
Cette fonction est déclarée de la façon suivante : *void compteur (void) ;*. A chaque front montant, cette fonction sera déclenchée, ce qui permet de compter les fronts.
Ex : moteur_A2.rise(&compteur2) ; // Exécution de la fonction compteur2 lorsqu'un front montant est détecté sur l'entrée moteur_A2

La fonction ***toggle_V*** est la fonction qui va être exécutée à intervalle régulier grâce au *ticker*. Elle permet de réaliser l'asservissement. Dans cette fonction pour chacun des moteurs, on calcule l'erreur entre le nombre de fronts montants mesuré et la commande. On calcule aussi l'erreur cumulée. On vient

ensuite appliquer le régulateur PI pour définir le nouveau rapport cyclique. Les erreurs sont affichées afin de vérifier l'asservissement. Les compteurs sont remis à 0. Cette fonction est réalisée toutes les 0,5s grâce à la commande suivante dans le *main* : *toggle_V_ticker.attach(&toggle_V, 0.5) ; // Exécution de la fonction toggle_V toutes les 0,5 s*
Cette fonction est déclarée en amont de la fonction *main*.

La fonction ***main*** est la fonction principale du code. Dans cette fonction, on vient initialiser les moteurs :
Ex : moteur_EN.period_ms(10) ; // Initialisation de la période à 10ms
moteur_EN.write(0) ; // Initialisation du rapport cyclique à 0, moteur à l'arrêt

On vient aussi exécuter l'asservissement comme expliqué précédemment. Enfin, dans une boucle `While(true)`, qui se répète indéfiniment, on réalise les fonctions permettant la communication. Cette partie sera détaillée ultérieurement. Dans cette boucle, on vient aussi choisir le sens de rotation des roues en agissant sur le rapport cyclique des sorties *moteur2_A* et *moteur3_A*. Si le rapport cyclique est à 0, alors les deux moteurs avancent.
Ex : moteur_2A.write(0) ; // Fait avancer le moteur 2

2.2 Interface graphique

Nous avons eu besoin de créer une interface graphique fonctionnelle permettant de communiquer avec le robot Véronica. Deux choses ont ainsi été importantes : la création de l'interface graphique et la communication filaire avec la carte nucléo qui communique en bluetooth avec le robot.

Tout d'abord, nous avons crée l'interface graphique avec *Tkinter*. L'interface est découpée en plusieurs zones : une zone d'entrée des données notamment la vitesse voulue pour le robot, la distance à parcourir ou l'angle de rotation ; une zone graphique pour, à termes, observer les variations de température et d'humidité au cours du temps (figure 6).

Il suffit ainsi de cliquer sur "Valider" pour envoyer au robot. Le fait de cliquer sur valider lance en effet une fonction grâce à la commande *tk.Button(self,text = "Valider",fg = "green",command = self.toto)* qui permet ainsi d'envoyer au robot (code complet en annexe).

Plus précisemment, on envoie la commande à une carte nucléo connectée à l'ordinateur qui l'envoie ensuite via un émetteur sans fil au robot. On a donc besoin d'une fonction qui prend en argument une chaîne de caractère correspondant à un entier à envoyer à la carte et qui envoie cet entier. On utilise ainsi la bibliothèque *pyserial* de python pour communiquer avec la carte nucléo. On définit tout d'abord une liaison série grâce à la fonction *serNuc = Serial('COM'+str(selectPort), 115200)* puis on peut ensuite transmettre des données en bytes à la carte grâce à la commande

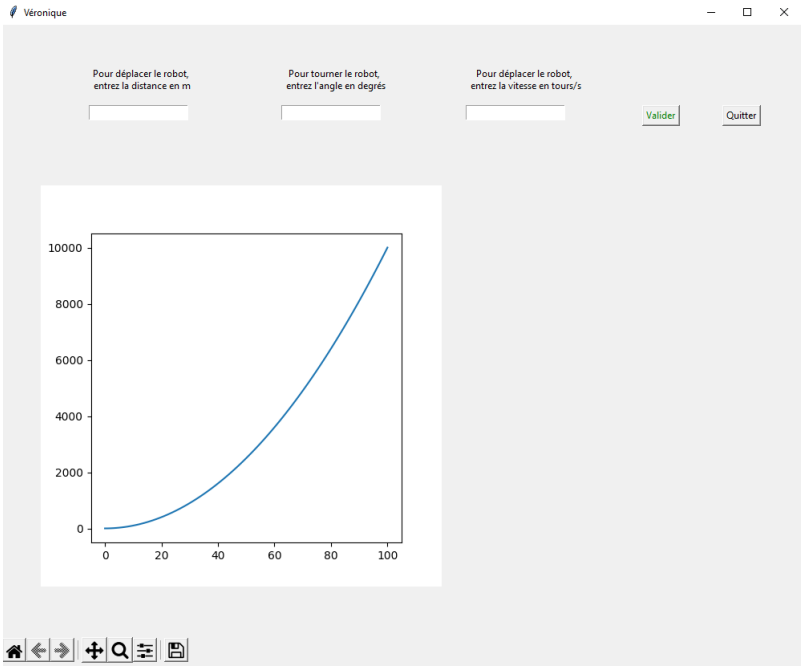


FIGURE 6 – Interface graphique

serNuc.write(). Cependant, il est donc important de convertir la chaîne de caractère qu'on a communiquer au code en octet transmissible à la carte. La fonction prend donc en argument une chaîne de caractère correspondant à un entier *data to send* et effectue :

```
int_to_send = int(data_to_send)
serNuc.write(int_to_send.to_bytes(1,"big"))
```

La spécification "big" est simplement une manière de coder les octets. On choisit ici "big" pour tout le projet par soucis d'homogénéité. On a ainsi crée une fonction qui peut être utilisée avec l'interface graphique pour communiquer avec la carte nucléo liée au PC.

2.3 Communication bluetooth

2.3.1 Présentation du module nRF24L01

Afin d'établir une connexion sans fil et garantir au robot une autonomie complète de déplacement, on utilise un module de communication `nRF24L01`. En en reliant un à la carte nucléo sur le robot, et un autre à l'ordinateur, il est possible d'assurer un échange de données en temps réel. Le cablâge suivant est donné à titre indicatif. Il est possible de choisir d'autres branchement pour la liaison SPI. Dans ce cas là, il convient de modifier les variables du codes pour qu'elles correspondent branchements faits.

L'alimentation du module doit être comprise entre 1,9 V et 3,3 V. Cependant, il est à noter que lors des émissions de données les pics de courant peuvent faire chuter

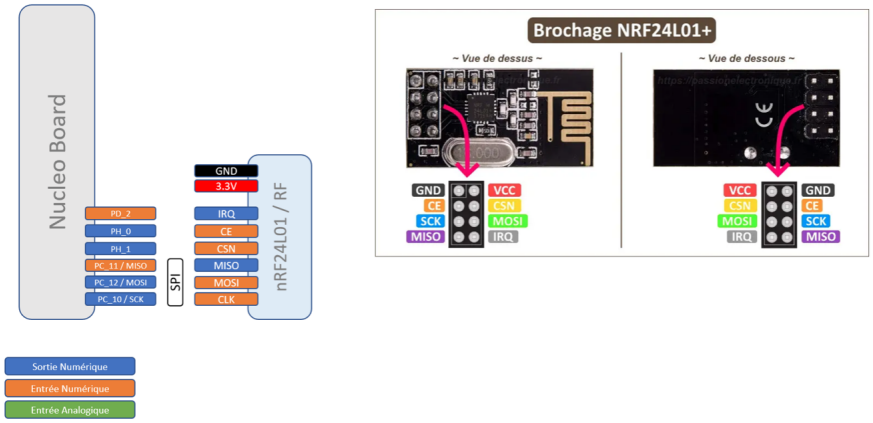


FIGURE 7 – Câblage du module de communication sans fil `nRF24L01`

la tension. Pour palier à ce problème, nous recommendons d'utiliser un condensateur de 10 μF afin de lisser la tension, celui-ci agissant comme un filtre passe-bas.

2.3.2 Rôles des différentes broches du module nRF24L01

Les différentes broches ont des rôles bien spécifiques qui interviennent dans le code fourni en annexe de ce document.

BROCHE	SIGNIFICATION	RÔLE
VCC	–	Alimentation du module (1,9 et 3,6 V)
GND	–	Alimentation du module (0 V)
SCK	Serial ClocK	Horloge communication SPI
MISO	Master In Slave Out	Voie de communication nRF24L01 -> NUCLEO
MOSI	Master Out Slave In	Voie de communication NUCLEO -> nRF24L01
CE	Chip Enable	Active le mode RX ou TX (émission / réception)
CSN	Chip Select Not	Active le module nRF24, lorsque mis à la masse (port SPI)
IRQ	Interrupt ReQuest	Permet d'interagir avec un microcontrôleur, si besoin

FIGURE 8 – Rôles des différentes broches du module `nRF24L01`

2.3.3 Principe de la communication sans fil

Il est possible de faire communiquer chaque module nRF24L01 entre eux. Pour cela, il faut établir une connexion entre deux modules. Les deux doivent être réglés sur la même fréquence de communication qui peut varier entre 2,4 et 2,5 MHz. Ensuite, il faut placer l'un en mode récepteur et l'autre en mode émetteur et ouvrir un canal de communication appelé "pipe" en anglais. Il s'agit en fait d'un tunnel

dans lequel circule l'information dans un sens ou dans l'autre suivant les modes réception/émission activés sur les modules.

Les modules nRF24L01 n'ont que 6 "pipes" de disponibles, dont seul le "pipe 0" peut être en mode réception ou émission. Les autres ne peuvent être utilisés que dans un des deux modes. Lorsque le module est mode réception, une simple commande `.read` permet de lire le message reçu. Inversement en mode émission, une simple commande `.write` permet d'écrire et d'envoyer un message.

Notons que la distance à laquelle les modules peuvent communiquer dépend aussi de la vitesse de communication. Plus cette vitesse est grande et plus la distance de communication est faible.

C'est donc grâce à ce module, que l'on pourra dans un premier temps, communiquer au robot les information concernant son déplacement ; et dans un second temps, recevoir les données collectées par le capteur d'humidité et température décrit dans la sous-partie suivante.

2.4 Système de détection en humidité et température

2.4.1 Présentation générale

Le système Temp&Hum MIKROE-4306 utilise un capteur intégré pour effectuer des mesures en température et humidité. Le capteur traite en intégralité les acquisitions et fait de manière automatique la conversion analogique-numérique. L'utilisateur n'a besoin que de se connecter au capteur pour effectuer les mesures et les lires.

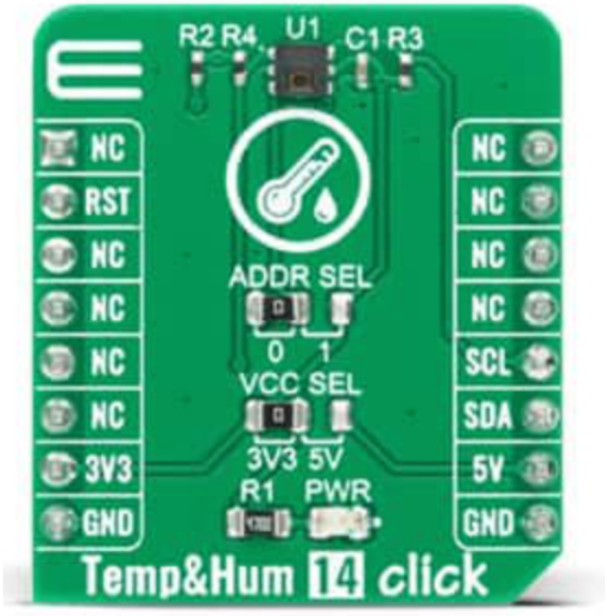


FIGURE 9 – Photo du système de mesure Temp&Hum MIKROE-4306

Le système emploie le protocole I2C pour réaliser le transfert des données numériques vers l'utilisateur. La connexion au capteur doit donc se faire en conséquence de cause, et il peut être nécessaire d'utiliser une librairie spécifique au capteur sur le micro-contrôleur utilisé.

2.4.2 Précision et tolérance du système de mesure

Les précisions des mesures sont données à 2% RH et 0,2°C près. Le capteur possède aussi une dérive intrinsèque de l'ordre de 0,04°C par an. Dans les conditions d'utilisation présentes, c'est-à-dire avec un capteur neuf, la dérive intrinsèque et négligeable.

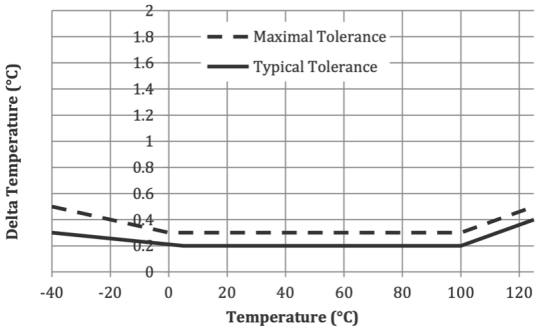


Figure a. Précision en température du capteur HTU31V RH/T SENSOR IC

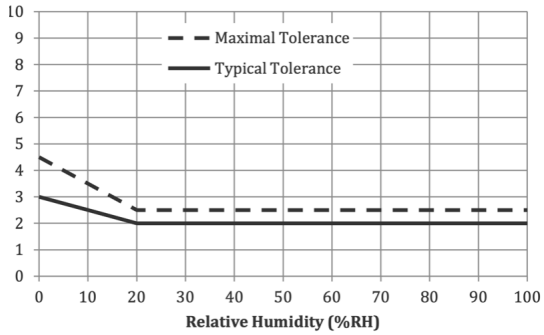


Figure b. Précision en humidité du capteur HTU31V RH/T SENSOR IC à 25°C

FIGURE 10 – Précision du capteur HTU31V RH/T SENSOR IC

La conversion analogique-numérique faite par le capteur induit également des erreurs et s'ajoute aux incertitudes de la mesures. Le paramètre osrRH0/osrT0 détermine l'incertitude sur la conversion. Il peut prendre 4 valeurs binaires : 00, 01, 10, 11. Le tableau ci-dessous donne les incertitudes associées suivant la valeur de ces deux paramètres.

Abbreviation	Binary value	Description
osrRH1..osrRH0	11	Humidity OSR = 3 (0.007%RH)
	10	Humidity OSR = 2 (0.010%RH)
	01	Humidity OSR = 1 (0.014%RH)
	00	Humidity OSR = 0 (0.020%RH)
osrT1..osrT0	11	Temperature OSR = 3 (0.012°C)
	10	Temperature OSR = 2 (0.016°C)
	01	Temperature OSR = 1 (0.025°C)
	00	Temperature OSR = 0 (0.040°C)

FIGURE 11 – Précision du détecteur suivant la valeur du paramètre OSR utilisé.

Ces erreurs restent marginales par rapport au reste mais elles sont tout de même données à l'attention de l'utilisateur.

2.4.3 Branchements et notes importantes.

Le système propose deux entrées d'alimentations, une à 3,3 V et l'autre à 5 V. Pour des raisons pratiques, nous utilisons l'alimentation à 3,3 V. Le protocole de branchement et d'acquisition des données décrit ci-dessous est fait avec une carte NUCLEO-L476RG. Si cette carte est différente, il convient de se reporter à la documentation dudit micro-contrôleur.

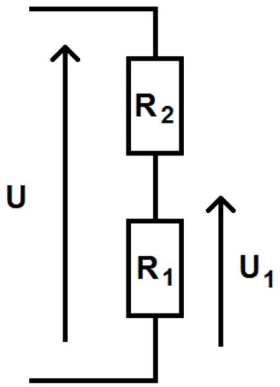


FIGURE 12 – Montage de pont diviseur en tension pour obtenir une tension donnée.

La batterie délivre une tension de V_B. Pour alimenter le capteur avec la tension voulue, il suffit de réaliser un pont diviseur en tension avec des résistances. Les deux tensions sont alors liées par la relation

$$U_1 = U \cdot \frac{R_1}{R_1 + R_2}$$

On récupère alors la tension d'alimentation voulue entre la résistance R_1 et R_2. Il est important d'obtenir cette tension puisque le système entier, détection et pilotage du robot ne possède qu'une seule et unique batterie.

Les entrées analogiques A4 et A5 de la carte NUCLEO-L476RG sont spécifiques au protocole I2C. Les autres ports d'entrées ne sont pas utilisables dans le cas présent. Pour un autre type de carte NUCLEO ou de micro-contrôleur, l'utilisateur doit s'assurer de la compatibilité I2C du port d'entrée de la carte et faire les modifications adéquates dans la définition des variables dans le code. Le signal SCL correspond aux données en température du capteur et le signal SCA donne les valeurs relevées pour l'humidité.

Pour que la carte NUCLEO puisse lire le signal analogique donné par la connexion I2C du détecteur, il faut qu'elle ait une librairie compatible avec le protocole I2C. En temps normal, cette librairie est déjà incorporée dans la carte. Dans le cas contraire, il faut se référer au site du fabriquant qui lui propose une librairie adaptée. Pour pouvoir utiliser les différentes sorties, il faut définir ces entrées I2C dans le code.

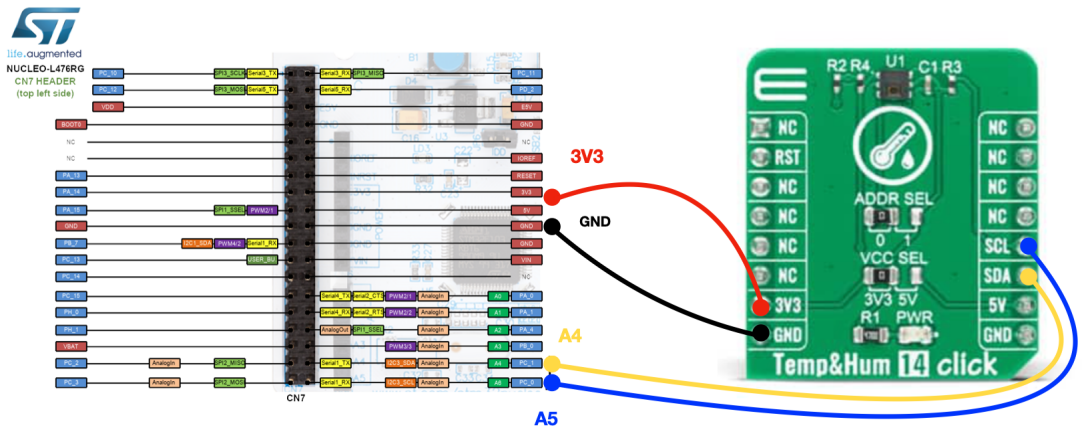


FIGURE 13 – Branchement entre le détecteur MIKROE-4306 et la carte NUCLEO L476RG.

3 Bilan du projet

Cette section a pour but d'évaluer la quantité de travail réalisé et les compétences acquises au cours du projet.

3.1 Bilan technique

Concernant l'aspect technique du robot Veronica, de multiples fonctions ont été réalisées comme cela a été expliqué dans les sections précédentes. En revanche, certaines fonctionnalités n'ont pas été réalisées par manque de temps et/ou d'organisation tandis que certaines fonctions ont été réalisées séparément mais n'ont pas encore été intégrées au prototype, c'est le cas de la mise en place du capteur de température et d'humidité. Au final, le travail réalisé est regroupé dans le tableau qui suit.

Fonction	Terminé / Non terminé (X / ✓)
Avancer en ligne droite	✓
Réaliser un asservissement en vitesse	✓
Réaliser une interface graphique	✓
Communiquer à distance à intervalle de temps régulier	✓
Contrôler la vitesse à distance	✓
Alimentation externe (autonomie)	✓
Tourner	X
Réaliser un asservissement en position	X
Réaliser et restituer des mesures	X

TABLE 2 – Bilan des fonctionnalités réalisées sur le robot Veronica

3.2 Bilan de compétences

Ce projet a permis de mettre en place des connaissances que l'on a acquis tout au long de notre scolarité. Aussi, cela a permis de développer de multiples compétences particulièrement importantes lorsque l'on entrera dans la vie active.

D'une part, le projet a permis de développer des compétences techniques :
- Prise en main de systèmes électroniques embarqués
- Mise en place de manips
- Rédaction de protocoles et de compte-rendus

D'autre part, il a aussi permis de développer des compétences transverses :
- Travail en équipe - Répartition des tâches
- Travail en autonomie
- Gestion d'un planning et des priorités
- Capacité à communiquer

3.3 Retour d'expérience de l'équipe

Il a été particulièrement intéressant de se mettre en situation de projet par équipe sur court-terme. En effet, nous n'avons pas encore eu l'occasion de nous mettre dans de telles conditions qui se rapprochent de notre future vie au travail. Cela nous a permis aussi de nous rendre compte qu'il n'était pas forcément facile de s'organiser pour travailler et faire un suivi du travail. Le résultat reste plutôt positif au vu du travail réalisé et de l'ambiance de l'équipe.

4 Annexes

4.1 Code de l'interface graphique et de la fonction de communication

Tout d'abord, voici le code de la fonction qui permet de communiquer avec la carte nucléo liée au PC :

```
from serial import Serial
import serial.tools.list_ports

def data_to_send(selectPort,data_to_send):
    serNuc = Serial('COM'+str(selectPort), 115200)  # Under Windows only
    appOk = 1
    while appOk:
        if data_to_send == '0' or data_to_send == '0':
            appOk = 0
        else:
            int_to_send = int(data_to_send)
            serNuc.write(int_to_send.to_bytes(1,"big"))
            while serNuc.inWaiting() == 0:
                pass
            data_rec = serNuc.read(1)  # bytes
            return(int.from_bytes(data_rec,"big"))
    serNuc.close()
```

Ensuite, on importe cette fonction dans le code de l'interface graphique pour l'utiliser d'où :

```
#Importation de base de fonctions

import tkinter as tk
from connexion import *
from serial import Serial
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)



class Veronique(tk.Tk):
    Data_distance = []
    Data_angle = []
    Data_vitesse = []

    def __init__(self):
```

```
        tk.Tk.__init__(self)
        self.geometry("1000x800")
        self.creer_widgets()
        self.plot()

    def toto(self):
        distance = self.entree_distance.get()
        angle = self.entree_angle.get()
        vitesse = self.entree_vitesse.get()
        print(vitesse)
        vitesse_fm_s = int(float(vitesse)*25);
        print(vitesse_fm_s)
        data_to_send(5,vitesse_fm_s)
        print("distance = "+distance + " cm")
        print("angle = "+angle + "°")
        print("vitesse = "+vitesse+" tours/s")

    def creer_widgets(self):
        self.title("Véronique")

        self.bouton = tk.Button(self, text="Quitter", command=self.destroy)
        self.bouton.place(x = 900,y = 100 )

        self.distance = tk.Label(self,text = "Pour déplacer le robot,\n entrez la d
        self.distance.place(x = 110 ,y = 50)

        self.entree_distance = tk.Entry(self)
        self.entree_distance.place(x = 110, y = 100)

        self.angle = tk.Label(self,text = "Pour tourner le robot,\n entrez l'angle
        self.angle.place( x = 350 ,y= 50)

        self.entree_angle = tk.Entry(self)
        self.entree_angle.place( x = 350, y = 100)

        self.vitesse = tk.Label(self,text = "Pour déplacer le robot,\n entrez la vi
        self.vitesse.place( x = 580 ,y= 50)

        self.entree_vitesse = tk.Entry(self)
        self.entree_vitesse.place( x = 580, y = 100)

        self.valider = tk.Button(self,text = "Valider",fg = "green",command = self.
        self.valider.place(x = 800, y= 100)

    def plot(self):
        fig = Figure(figsize = (5,5),dpi = 100)
```

```
        y = [i**2 for i in range(101)]
        plot1 = fig.add_subplot(111)
        plot1.plot(y)
        self.canvas = FigureCanvasTkAgg(fig,master = self)
        self.canvas.draw()
        self.canvas.get_tk_widget().place(x = 50,y = 200)

        self.toolbar = NavigationToolbar2Tk(self.canvas,self)
        self.toolbar.update()
        self.toolbar.pack()



if __name__ == "__main__":
    app = Veronique()
    app.mainloop()
```