Projet 2020_B002 : Robot Veronica

Rapport Technique

Auteurs: FRESLIER Clément, GRANVEAU Gabriel, HENAFF Johan

Table des matières

I. Présentation du projet

1. Mise en contexte

Dans le cadre de son programme d'exploration de Mars, la société SOLEC a pour projet le développement d'un robot motorisé afin de pouvoir détecter l'éventuelle présence d'eau sur la planète.

Ce système doit pouvoir être guidé à distance depuis une base terrienne en suivant un trajet composé de tronçons de lignes droites et de virages. Il doit de plus relever, à intervalles réguliers, la température et le taux d'humidité, afin de les transmettre au centre d'opérations pour analyse.

2. Cahier des charges

Afin de pouvoir remplir sa fonction principale, le robot doit obéir à un cahier des charges précis:

Il est important que le robot puisse obéir à des commandes de position et d'angle lui permettant de se déplacer en ligne droite ou d'effectuer des rotations sur lui-même. Ces commandes doivent être envoyées à distance.

Afin de pouvoir effectuer les mouvements imposés par l'opérateur, la structure suivante a été retenue:

- deux roues motorisées indépendamment
- une roue libre

De plus, le robot doit être capable d'effectuer des mesures de température et d'humidité tous les 10cm, et les transmettre au centre d'opérations toutes les 10 minutes.

Pour cela, il est important de mettre au point des capteurs permettant d'effectuer des mesures dans l'environnement martien, défini avec les contraintes suivantes :

- Température minimale : 150 K (-123°C)
- Température maximale: 293K (20°C)
- Poussières en suspension

Les performances minimales attendues par le robot sont les suivantes :

- Le robot doit pouvoir avancer à une vitesse entre 10 et 30 cm/s
- Une erreur maximale de 2 cm sur la position et 3° sur l'angle est tolérée
- Le robot doit pouvoir effectuer un parcours d'1km sans recharger ses batteries
- L'interface Homme-Machine doit être la plus ergonomique possible, au sens où aucune compétence informatique spécifique ne doit être requise pour son utilisation.
- Les données doivent pouvoir être affichées en fonction du temps ou de la distance selon le choix de l'opérateur

3. Répartition des tâches

Au regard du schéma fonctionnel (**figure 1**), le projet est découpé en quatres parties.

- Interface
- Communication
- Moteur
- Capteur

La partie interface s'occupera de l'interface homme-machine et de centraliser toutes les données arrivant des autres parties. La partie communication traitera d'établir la liaison à distance entre le robot et l'utilisateur. La partie moteur sera en charge de choisir et d'étudier la motorisation du robot. Enfin, la partie capteur étudiera les capteurs de température et d'hygrométrie qui seront implémenté sur le robot. En raison de la crise du COVID-19 ces parties seront traité indépendamment puis réunies ultérieurement. De plus étant donné que notre équipe n'est constitué que de trois membre, la partie capteur n'a pas pu être traité dans son intégralité par manque de temps.

4. Schéma fonctionnel

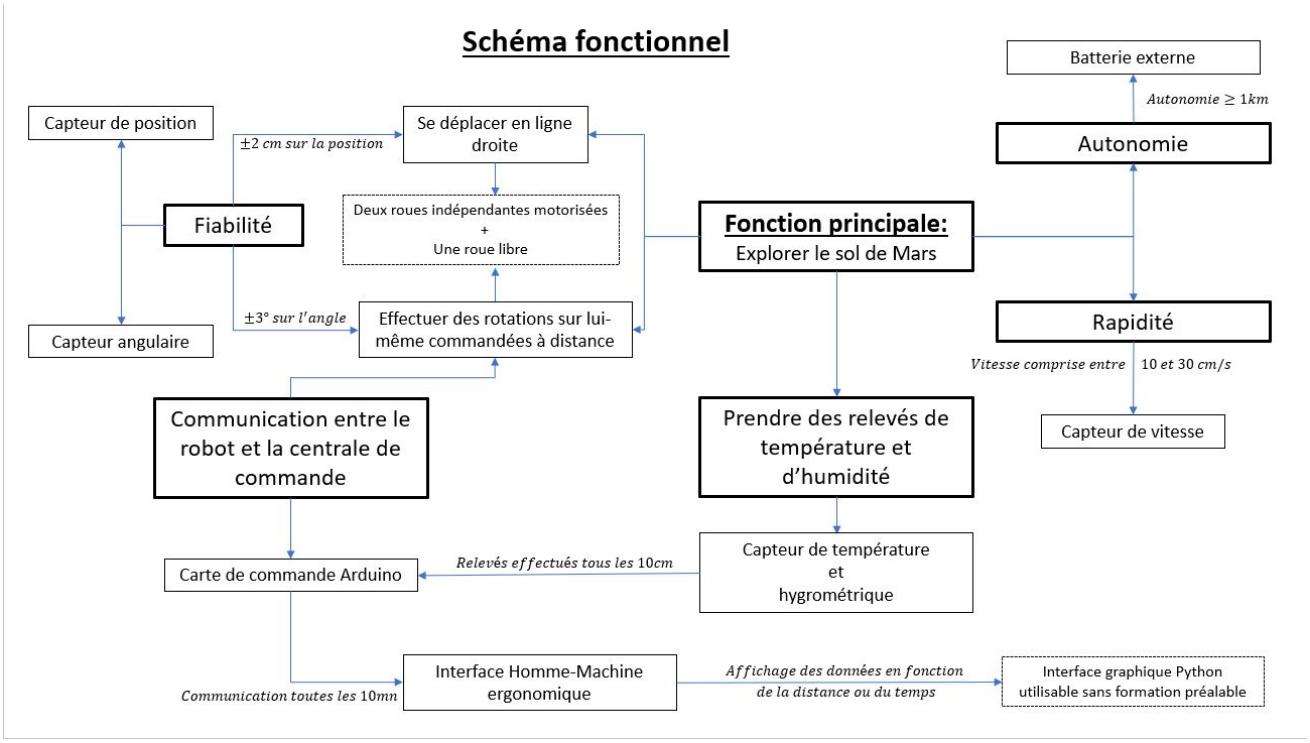
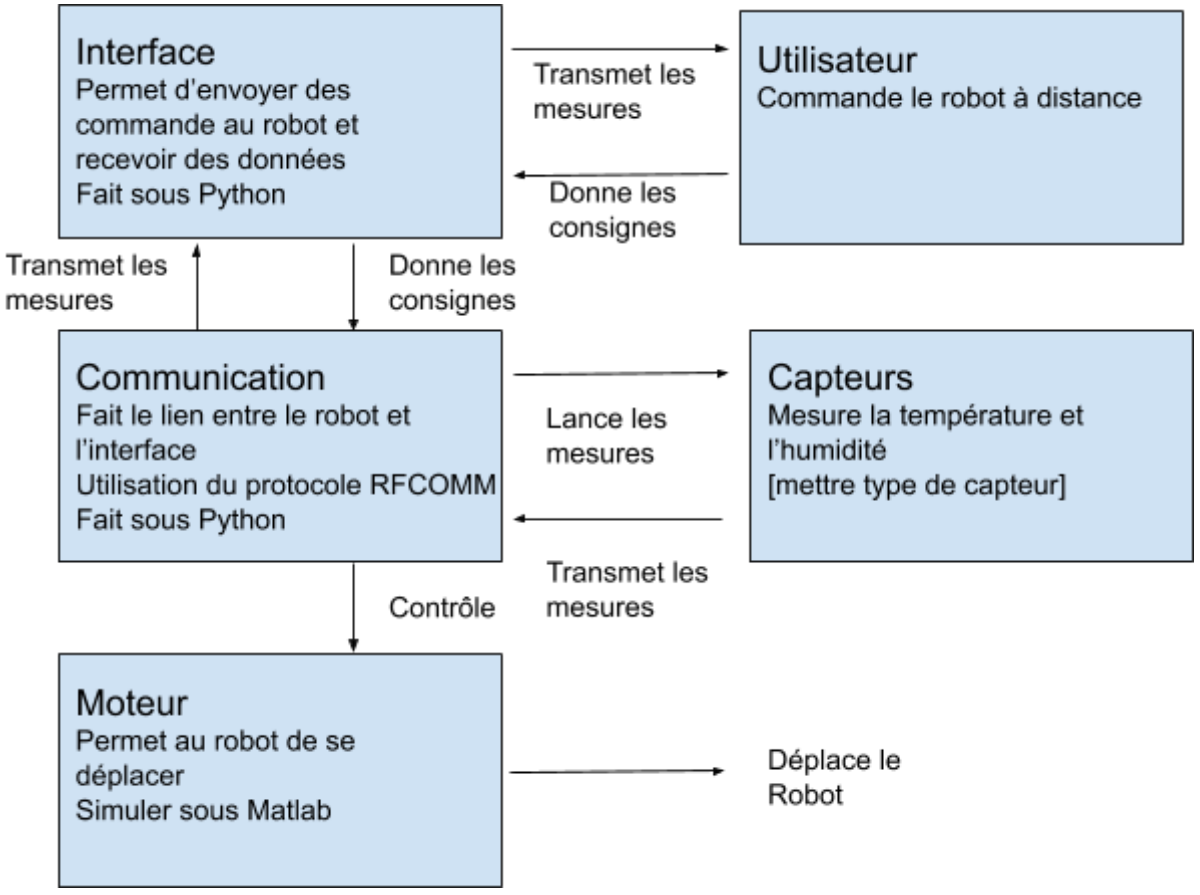


Figure 1: schéma de l'architecture et découpage fonctionnel du robot Veronica



- Découpage fonctionnel système

II. Interface Homme-Machine

1. Présentation générale

La communication entre le robot Veronica et l'utilisateur doit se faire le plus ergonomiquement possible, conformément au cahier des charges proposé.
C'est pourquoi il a été nécessaire de développer une interface utilisateur simple d'utilisation, permettant d'accéder à un ensemble de fonctionnalités utiles pour le pilotage à distance du robot ainsi que l'acquisition et la visualisation des données.
L'interface proposée permet donc de

- Rentrer des commandes de position, de vitesse et de vitesse angulaire qui seront ensuite transmises au robot
- Récupérer les données collectées par le robot et les afficher en fonction de la date d'acquisition ou de la position par rapport à une position de référence
- Afficher sur un graphique la position et le taux d'humidité en fonction de la date d'acquisition
- Afficher les valeurs moyennes
- Enregistrer les données dans un fichier texte

L'interface a également été pensée pour être la plus simple d'utilisation possible, notamment grâce à un système de fenêtres multiples permettant de ne pas surcharger l'interface.

Les sections suivantes présenteront plus en détail le choix de la structure du programme, les différentes fonctionnalités, ainsi que les difficultés techniques rencontrées lors du développement.

Des copies d'écran du code commentées illustreront chaque partie lorsque nécessaire et le code complet est fourni dans une annexe au format .py.

2. Structure du programme

Pour réaliser cette interface, nous avons choisi d'utiliser le langage Python, notamment car il s'agit d'un langage de programmation haut niveau, et donc facile à prendre en main, mais aussi car il est très documenté et possède de nombreux modules intégrés ou téléchargeables permettant de grandement simplifier le travail du codeur.
Ce choix est mutualisé entre l'interface graphique et la communication Bluetooth, ce qui permet de pouvoir facilement récupérer les données d'un programme pour l'affecter à l'autre.
L'interface a été réalisée et fonctionne sur Python 3.8.2.
Elle n'a pas été testée sur des versions antérieures, et son fonctionnement ne peut être garanti, surtout sur les versions Python 2.x.x.
Le module principal utilisé dans cette application est tkinter, et permet la création d'interfaces graphiques.
D'autres modules ont dû être importés pour le bon fonctionnement du programme, et sont listés sur l'image ci-dessous :

```
'''Libraries
--------------------------------------------------------------------------------'''
import os
from functools import partial
from PIL import ImageTk, Image
from datetime import datetime, timedelta
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
import matplotlib.animation as animation
from time import sleep
from matplotlib import style


try:
    # Python 2
    import Tkinter as tk
    from Tkinter.messagebox import *
    import Tkinter.ttk as ttk

except ImportError:
    # Python 3
    import tkinter as tk
    from tkinter.messagebox import *
    import tkinter.ttk as ttk
```

Figure 2: Capture d'écran des fonctions et modules importés pour l'IHM

Pour le développement du programme, une structure utilisant la programmation orientée objet a été privilégiée, car elle permet une plus grande liberté d'implémentation au moyen de la création de classes personnalisées, possédant les attributs et méthodes voulues par le développeur.

La fonctionnalité d'héritage permet également de pouvoir créer une classe « enfant » sur le modèle d'une classe « parent », et donc d'avoir accès aux attributs et méthodes de cette classe « parent », ce qui est très utile.

La structure du code source est donnée ci-dessous.

Figure 3: Structure du code source, en bleu les modules importés, en rouge les classes et en noir les méthodes et fonctions

Il est de plus nécessaire de pouvoir récupérer et traiter des données externes au programme en temps réel, ainsi que d'envoyer des données en temps réel.

Pour cela, il a été décidé d'utiliser la programmation parallèle (ou multithreading).

La fonction Retrieve_Data s'occupant de récupérer les données envoyées par le robot est détaillée dans la section dédiée.

On donne sur la figure ci-dessous le schéma de principe de l'interface légendé. La classe HMI_Veronica est la classe principale de l'ensemble du programme.

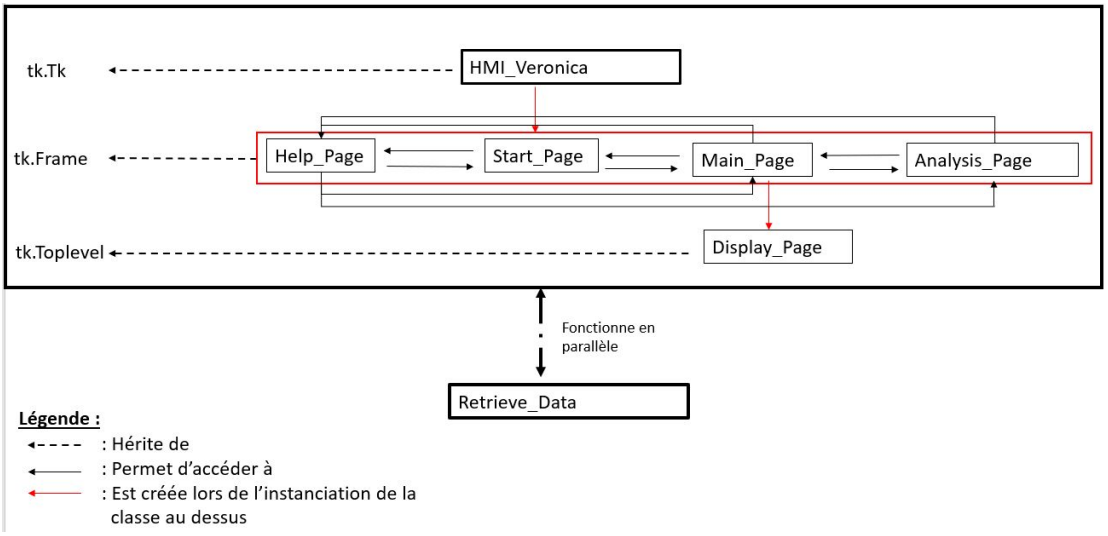


Figure 4: schéma de principe de la structure de l'interface

Afin d'instancier la classe HMI_Veronica, il suffit de créer un objet de cette classe. Une fois la classe instanciée, les pages héritant de **tk.Frame** vont être créées et la page définie par **Start_Page** est élevée au premier plan.

Il est possible de changer la page en premier plan selon les relations fléchées.

De plus, la classe **Display_Page** est automatiquement instanciée lors de la première mise au premier plan de la page définie par **Main_Page**.

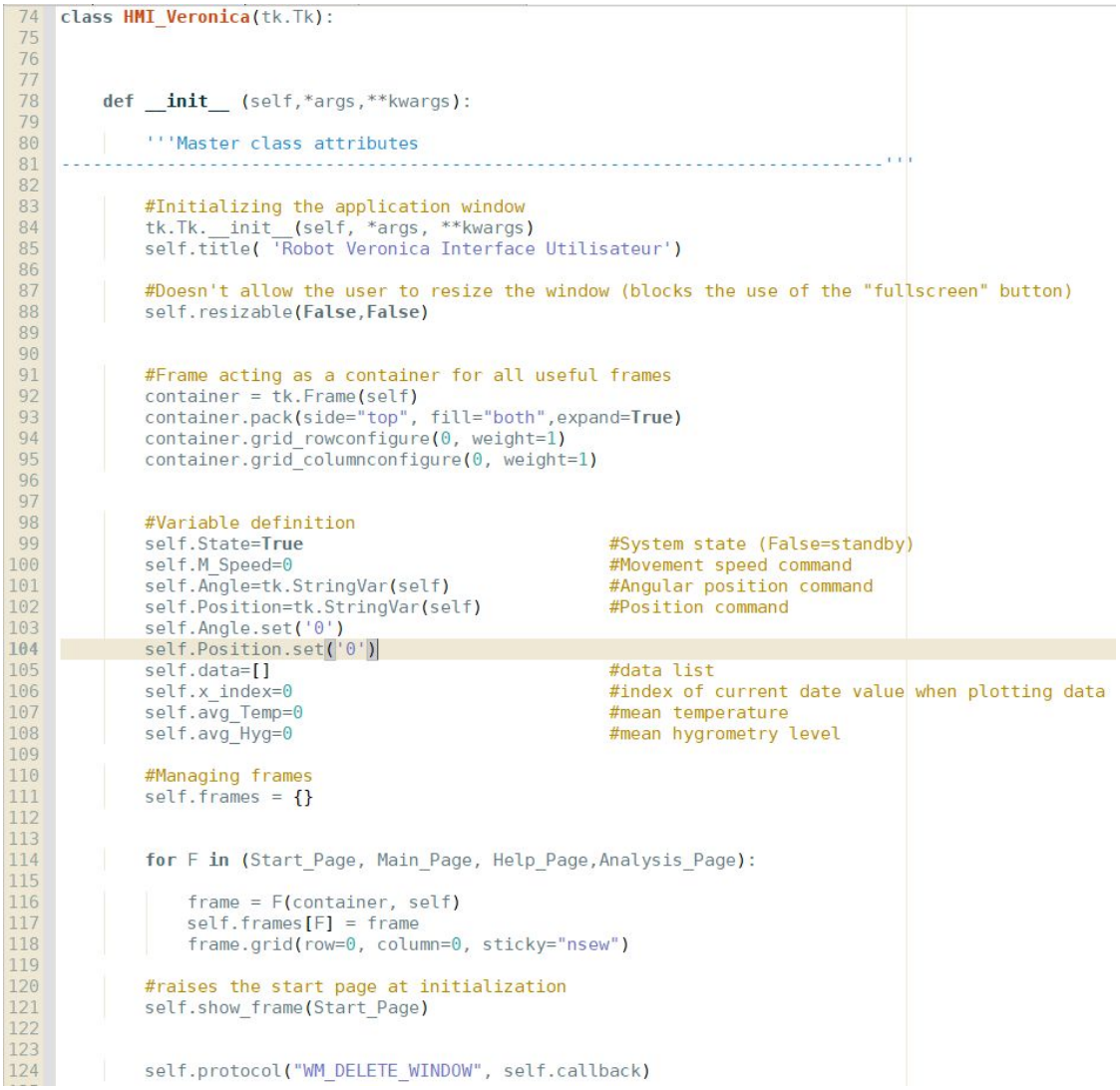
Le détail des différentes classes est donné dans la section suivante.

3. Détail du code

Dans cette section, nous aborderons le détail des différentes classes du programme et fonctionnalités de l'interface :

- **HMI_Veronica** : Il s'agit de la classe principale de l'application, c'est elle que l'on appelle pour créer l'interface.

  Ses attributs sont donnés sur la figure ci-dessous :



```
74  class HMI_Veronica(tk.Tk):
75
76
77
78      def __init__ (self,*args,**kwargs):
79
80          '''Master class attributes
81  ------------------------------------------------------------------------------'''
82
83          #Initializing the application window
84          tk.Tk.__init__(self, *args, **kwargs)
85          self.title( 'Robot Veronica Interface Utilisateur')
86
87          #Doesn't allow the user to resize the window (blocks the use of the "fullscreen" button)
88          self.resizable(False,False)
89
90
91          #Frame acting as a container for all useful frames
92          container = tk.Frame(self)
93          container.pack(side="top", fill="both",expand=True)
94          container.grid_rowconfigure(0, weight=1)
95          container.grid_columnconfigure(0, weight=1)
96
97
98          #Variable definition
99          self.State=True                             #System state (False=standby)
100         self.M_Speed=0                              #Movement speed command
101         self.Angle=tk.StringVar(self)               #Angular position command
102         self.Position=tk.StringVar(self)            #Position command
103         self.Angle.set('0')
104         self.Position.set('0')
105         self.data=[]                                #data list
106         self.x_index=0                              #index of current date value when plotting data
107         self.avg_Temp=0                             #mean temperature
108         self.avg_Hyg=0                              #mean hygrometry level
109
110         #Managing frames
111         self.frames = {}
112
113
114         for F in (Start_Page, Main_Page, Help_Page,Analysis_Page):
115
116             frame = F(container, self)
117             self.frames[F] = frame
118             frame.grid(row=0, column=0, sticky="nsew")
119
120         #raises the start page at initialization
121         self.show_frame(Start_Page)
122
123
124         self.protocol("WM_DELETE_WINDOW", self.callback)
```

Figure 5: Attributs de la classe HMI_Veronica

Parmi ces attributs, on trouve toutes les variables pertinentes à l'utilisation de l'application, telles que la position, la vitesse ou les données transmises par le robot.

On trouve de plus les commandes lignes 114 à 118 qui permettent, à l'instanciation de la classe HMI_Veronica, de créer toutes les pages utiles pour l'application en tant qu'objet tk.Frame, placées dans la frame « container ».

Cette classe possède de plus les méthodes suivantes :

- **show_frame** : permet de choisir la page à élever au premier plan à l'aide de la méthode **tkraise** inhérente à tkinter et de gérer la géométrie des pages.

Par défaut, la page définie par **Start_Page** est élevée au premier plan.

-**Shutdown** : permet de gérer la fermeture de l'application à l'aide des boutons « Arrêt Robot »

-**Callback** : permet de gérer la fermeture de l'application à l'aide de la commande Windows (croix rouge), et est appelée par la méthode native tkinter **protocol**, permettant de gérer les interactions entre l'utilisateur et le Windows manager.

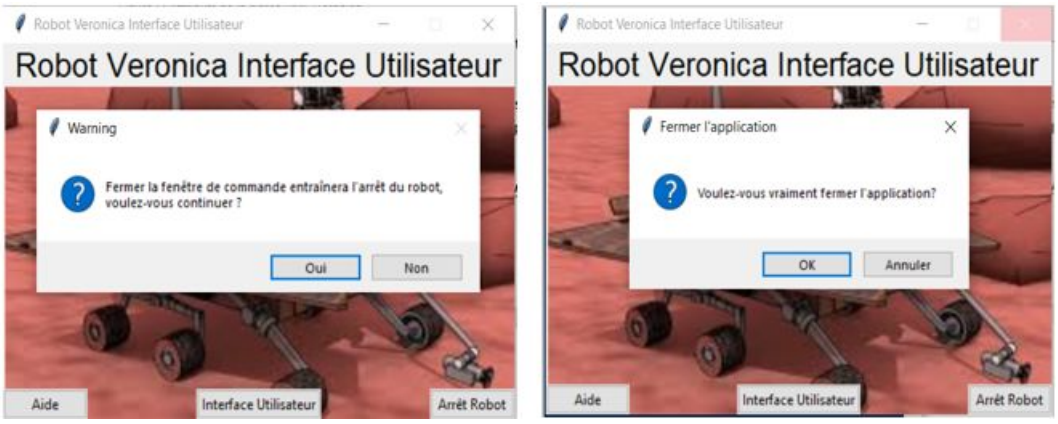


Figure 6: Fenêtre de fermeture du programme. A gauche : appel à la méthode "Shutdown", à droite : appel à la méthode "Callback"

- **Get_mode** : permet de récupérer le mode dans lequel on souhaite afficher les données au moyen de l'attribut **var** qui sera présenté dans la section **Main_Page**.

- **Start_Page** : Classe gérant le menu principal de l'application.

```
269 class Start_Page(tk.Frame):
270
271
272
273     def __init__(self, master, controller):
274
275         '''Class Attributes
276 ----------------------------------------------------------------------------'''
277
278         #Initializing the frame
279         tk.Frame.__init__(self,master,width=460,height=310)
280         self.master=master
281
282
283         #Start Page Background Image
284         self.bg_img=tk.Canvas(self,width=460,height=310)
285         self.img=ImageTk.PhotoImage(file=r"images\Robot_Veronica_Img.png")
286         self.background=self.bg_img.create_image(230,155,image=self.img,anchor="center")
287         self.bg_img.place(x=0,y=0)
288
289
290         #Main Page Title
291         self.label = ttk.Label(self, text="Robot Veronica Interface Utilisateur",font=('Helvetica',
    21),anchor='center')
292         self.label.place(x=230,y=0,anchor="n",relwidth=1)
293
294
295         #Main Page Menu : buttons allowing the user to navigate between help page and command page and quit
    application button
296         interface_button = ttk.Button(self, text="Interface Utilisateur",
297                             command=lambda: controller.show_frame(Main_Page))
298         interface_button.place(x=230,y=310,anchor="s")
299
300
301         help_button = ttk.Button(self, text="Aide",
302                             command=lambda: controller.show_frame(Help_Page))
303         help_button.place(x=0,y=310,anchor='sw')
304
305
306         exit_button=ttk.Button(self,text='Arrêt Robot',command=controller.Shutdown)
307         exit_button.place(x=460,y=310,anchor='se')
```

Figure 7: Classe Start_Page

Cette classe ne possède pas de méthode particulière, et appelle seulement les méthodes « **show_frame** » et « **Shutdown** » de la classe principale en tant qu'action à effectuer lorsque l'utilisateur appuie sur les boutons, représentés par des objets de type **tk.Button**.

Elle possède les attributs suivants :

-**title** : correspondant au nom de l'application

-**background** : une image de fond, libre de droit, disponible au lien suivant : https://www.blendswap.com/blend/15248, et stockée dans un Canvas tkinter

Le menu principal de l'application ressemble donc à ceci :

Figure 8: Menu principal de l'interface utilisateur du Robot Veronica

On peut, à partir de cette fenêtre, au choix accéder à la fenêtre d'aide, à l'interface utilisateur constituée de la fenêtre de commande (voir section **Main_Page**) et de celle d'affichage (voir section **Display_Page**), ou bien fermer l'interface.

- **Main_Page** : Classe gérant la fenêtre de commande de l'interface. Elle est mise au premier plan sur appui du bouton « Interface Utilisateur » ou « Retour à l'interface » sur les autres fenêtres.

```
323
324         #controller=master class(HMI_Veronica here), used to call methods inherited from that class, master=tk.Frame
325         self.controller=controller
326         self.master=master
327
328
329         #Display Page, None value means there isn't any display page created yet
330         self.Data_Page=None
```

```
361
362         #Setting Command Speed and validating entry. The set value is stored in M_Speed attribute (inherited from master class)
363         MS_Label=tk.Label(self,text="Commande de vitesse \n (Une commande entre 0 et 9 \n imposera une vitesse de 10m/s) ")
364         MS_Label.grid(row=2,column=0,pady=20)
365         self.MS_Slider=tk.Scale(self,from_=0,to=30,tickinterval=10,resolution=1,orient='horizontal') #slider, units in cm/s
366         self.MS_Slider.grid(row=2,column=1,pady=20)
367         MS_Set=tk.Button(self,text="Valider",command=self.Set_MSpeed)
368         MS_Set.grid(row=2, column=2, pady=20,padx=20)
369
370
371
372         #Setting Position Command and validating entry. The value is stored using Position.get() method
373         Position_Label=tk.Label(self,text="Commande de position (en cm)")
374         Position_Label.grid(row=4,column=0,pady=20)
375         Position_Entry=tk.Entry(self,textvariable=self.controller.Position)
376         Position_Entry.grid(row=4,column=1,padx=5,pady=20)
377         POS_Set=tk.Button(self,text="Valider",command= self.Set_Position)
378         POS_Set.grid(row=4, column=2, pady=20,padx=20)
379
380
381         #Setting Angular Command and validating entry. The value is stored using Angle.get() method
382         Angular_Label=tk.Label(self,text="Commande de position angulaire (en °)")
383         Angular_Label.grid(row=6,column=0,pady=20)
384         Angular_Entry=tk.Entry(self,textvariable=self.controller.Angle)
385         Angular_Entry.grid(row=6,column=1,padx=5,pady=20)
386         ANG_Set=tk.Button(self,text="Valider",command= self.Set_Angle)
387         ANG_Set.grid(row=6, column=2, pady=20,padx=20)
388
389
390         #Integer, used to decide wether the data will be displayed as a function of time or distance
391         self.var=tk.IntVar(master)
392         self.var.set(0)
393
394
395         #Setting the display mode
396         Display_choice_Label=tk.Label(self,text="Choix du mode d'affichage")
397         Display_choice_Label.grid(row=10,column=1,pady=20)
398         Display_choice_Label.config(font=("Courrier",30))
399         Display_choice_time=tk.Radiobutton(self,text='Temps',variable=self.var,value=1,command=partial(self.Update_Display,controller))
400         Display_choice_time.grid(row=11,column=1,pady=10)
401         Display_choice_distance=tk.Radiobutton(self,text='Distance',variable=self.var,value=2,command=partial(self.Update_Display,controller))
402         Display_choice_distance.grid(row=13,column=1,pady=10)
```

Figure 9: Attributs de la classe Main_Page. Seuls les attributs n'ayant pas été évoqués dans les sections précédentes sont présentés ici.

En plus de posséder des boutons de changement de fenêtre (se référer à la figure 7 pour voir la structure de ces boutons), cette classe est dotée des attributs suivants :
- **MS_slider** (objet **tk.Slider**) : permettant de choisir la commande de vitesse à imposer au robot
-**Position_Entry** et **Angular_Entry** (objet **tk.Entry**) : champs de saisie permettant d'entrer les consignes de position et d'angle.

-**POS_Set** et **Ang_Set** : boutons permettant de valider la saisie des consignes et d'attribuer les valeurs des consignes aux attributs de la classe principale correspondants, à savoir **Position**, **Angle** et **M_Speed**.
-**Display_choice_time** et **Display_choice_distance** (objets **tk.Radiobutton**) : permet également de choisir le mode d'affichage des données, en fonction du temps ou de la distance.
-**var :** objet de type **tk.IntVar** auquel on affecte, suivant que l'on souhaite afficher les données en fonction du temps ou de la distance la valeur 1 ou 2.

Une méthode de la classe principale permettant de récupérer la valeur de l'attribut **var** a été créée, afin de simplifier la récupération de cette valeur dans les autres classes dans lesquelles il doit être appelé.

Les méthodes de cette classe peuvent être classées dans deux catégories.
Tout d'abord, les méthodes permettant récupérer les valeurs entrées par l'utilisateur et les stocker dans les variables correspondantes ; il s'agit des méthodes « **Set_MSpeed** », « **Set_Position** » et « **Set_Angle** ».
La méthode « **Update_Display** » permet quant à elle, si la fenêtre d'affichage existe, d'actualiser l'affichage des données. Si cette fenêtre n'existe pas, alors elle est créée.

Cette fenêtre possède des boutons permettant d'ouvrir la fenêtre d'aide, le menu principal ou la fenêtre d'analyse des données, ainsi qu'un bouton pour fermer l'application.

- **Display_Page** : Cette classe gère les fonctionnalités d'affichages des données reçues par le robot en temps réel.

  Cette classe n'hérite pas de **tk.Frame** comme les autres sous-classes du programme, mais de la classe **tk.Toplevel**, qui permet l'affichage d'une seconde fenêtre, ce qui permet donc d'avoir deux fenêtres en parallèles, une pour commander le robot, l'autre pour visualiser les données reçues.

  Elle est créée automatiquement lorsque la page définie par Main_Page est élevée au premier plan, et envoyée en arrière-plan lorsqu'une autre fenêtre est appelée.

  Si l'utilisateur ferme la fenêtre en utilisant la commande Windows, cette page peut être recréée de plusieurs manières :

  -soit en envoyant en arrière-plan puis en remettant au premier plan la fenêtre de commande (avec n'importe quelle autre fenêtre)

  -soit en appuyant sur l'un des modes d'affichage (rappuyer sur le mode déjà sélectionné permet de recréer la fenêtre).

```
470 class Display_Page(tk.Toplevel):
471
472
473
474     def __init__(self,master,controller):
475
476         '''Class Attributes
477 ----------------------------------------------------------------------------'''
478
479         tk.Toplevel.__init__(self, master,bg='black')
480
481
482         #controller=master class, used to call methods inherited from that class
483         self.controller=controller
484         self.master=master
485
486         self.mode=controller.Get_mode()
487
488
489         #String to be displayed
490         self.string=""
491
492
493         #Number of elements to display
494         self.buffer_size=35
495
496         #Display Label
497         self.Data_Label=tk.Label(self,fg='green',bg='black')
498         self.Data_Label.place(x=0,y=0)
```

Figure 10: Attributs de la classe Display_Page

Les attributs propres à cette classe sont les suivants :

-**mode :** le mode d'affichage, obtenu au moyen de la méthode « **Get_mode** »

-**string :** une chaîne de caractère, initialement vide, permettant de stocker les données à afficher

-**buffer_size** : le nombre d'élément maximal à afficher sur la page. La valeur 35 à été choisie afin d'afficher des données sur l'ensemble de la page

-**Data_Label** : le Label (objet **tk.Label**) permettant d'afficher les données sur la fenêtre.

Cette classe possède de plus deux méthodes :

-**Display_Data** : permet d'afficher le texte contenu dans **string** sur la fenêtre

-**Update** : permet de mettre en forme **string** avec les données contenues dans **data** selon le mode d'affichage choisi.

Cette méthode permet également de mettre à jour le format de l'affichage des valeurs moyennes, qui sont affichées dans la page **Analysis_Page**.

Ces données sont obtenues en temps réel à l'aide de la fonction **Retrieve_Data** (voir section associée pour le détail du fonctionnement).

Une fois les données affichées, on change la valeur de **string** en tant que chaîne vide jusqu'au prochain appel de la méthode **Update**.

Le code de la méthode ainsi que les commentaires et la docstring associée est donné ci-dessous.

```
515
516     def Update(self):
517
518         '''Takes the data stored in self.controller.data and puts them into te string attribute, accordingly to the selected mode.
519            The number of data to display must be lower than the display limit, and is controlled by the local variable i.
520            This method also updates the mean temperature and hygrometry calculated in Retrieve_Data and displayed in the Analysis_Page class
521            Inputs: self
522            Outputs: None
523            Author: FRESLIER Clément, 26/04/2020                                    '''
524
525         #Number of displayed data
526         i=0
527
528         #Display mode
529         self.mode=self.controller.Get_mode()
530
531         #Data to be displayed, read from last element to first
532         for Position, Angle, Date, Temperature, Hygrometry in reversed(self.controller.data):
533
534             #if the number of displayed data is higher than limit
535             if i >= self.buffer_size:
536                 break
537
538             #if selected data mode is "time"
539             if self.mode==1:
540
541                 self.string+= "{0} : Température= {1}°C, Hygrométrie= {2}% \n".format(Date,Temperature,Hygrometry)
542
543             #if selected data mode is "position"
544             elif self.mode==2:
545
546                 self.string+="Postion: {0}cm, {1}° :Température= {2}°C, Hygrométrie= {3}% \n".format(Position, Angle, Temperature, Hygrometry)
547
548             i+=1
549
550         #update mean temperature and stores it the T_str attribute of Analysis_Page class
551         self.controller.frames[Analysis_Page].T_str= "Température Moyenne ={:.3f}°C".format(self.controller.avg_Temp)
552         self.controller.frames[Analysis_Page].H_str= "Hygrométrie Moyenne ={:.3f}%".format(self.controller.avg_Hyg)
553
554         #Call display methods to refresh the label's text
555         self.Display_Data()
556         self.controller.frames[Analysis_Page].Update_avg()
557
558         self.string=""
559
```

Figure 11: code de la méthode Update

La fenêtre de commande, associée à celle d'affichage ressemble donc à ceci :

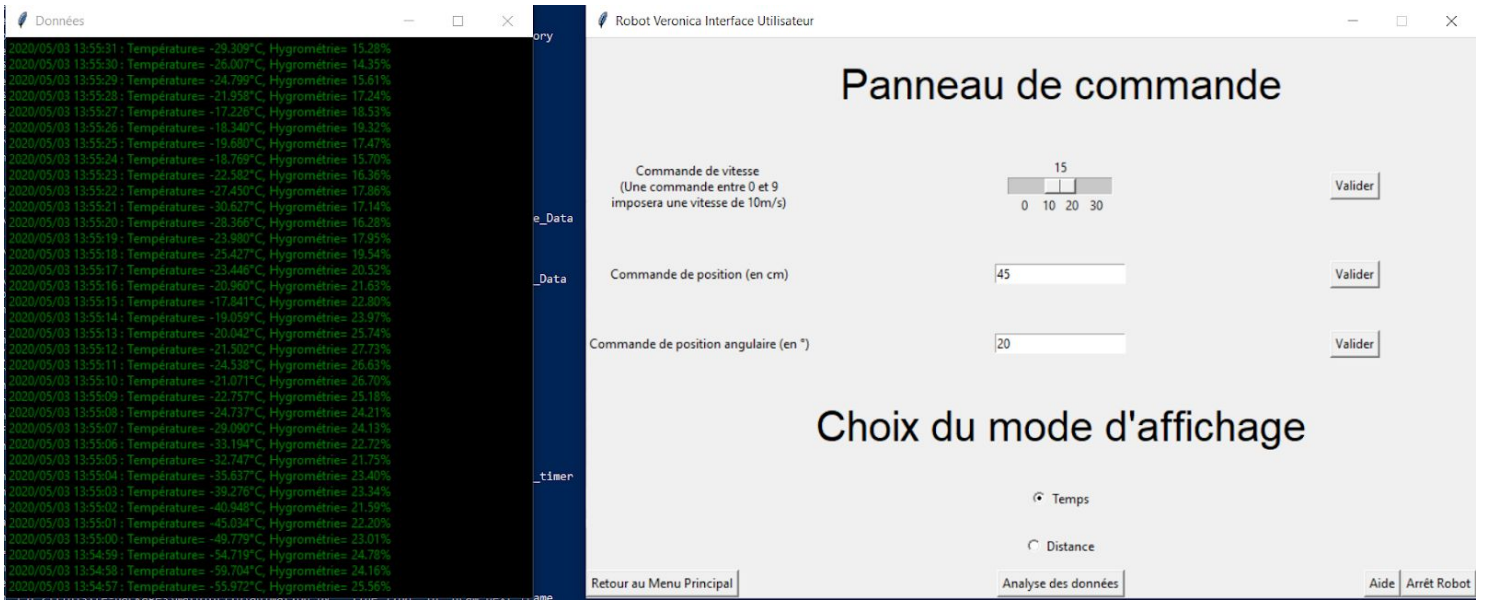


Figure 12: Panneau de commande et fenêtre d'affichage de l'interface utilisateur du Robot Veronica. Les données sont ici affichées en fonction du temps et ne correspondent pas aux valeurs des consignes arbitraires de position, angle et vitesse ont été données pour illustrer les fonctionnalités. La zone centrale en bleu est un extrait du Windows PowerShell utilisé pour lancer le programme en background.

- **Help_Page** : Cette classe gère la fenêtre d'aide à l'utilisation de l'interface utilisateur du Robot Veronica.
  Il s'agit d'une zone de texte expliquant le fonctionnement de l'interface, possédant une barre de défilement permettant à l'aide de la molette de la souris de faire défiler le texte.

  Cette fenêtre est également dotée de boutons permettant de rejoindre le menu principal ainsi que la fenêtre de commande.

  La figure ci-dessous montre un aperçu de la fenêtre :

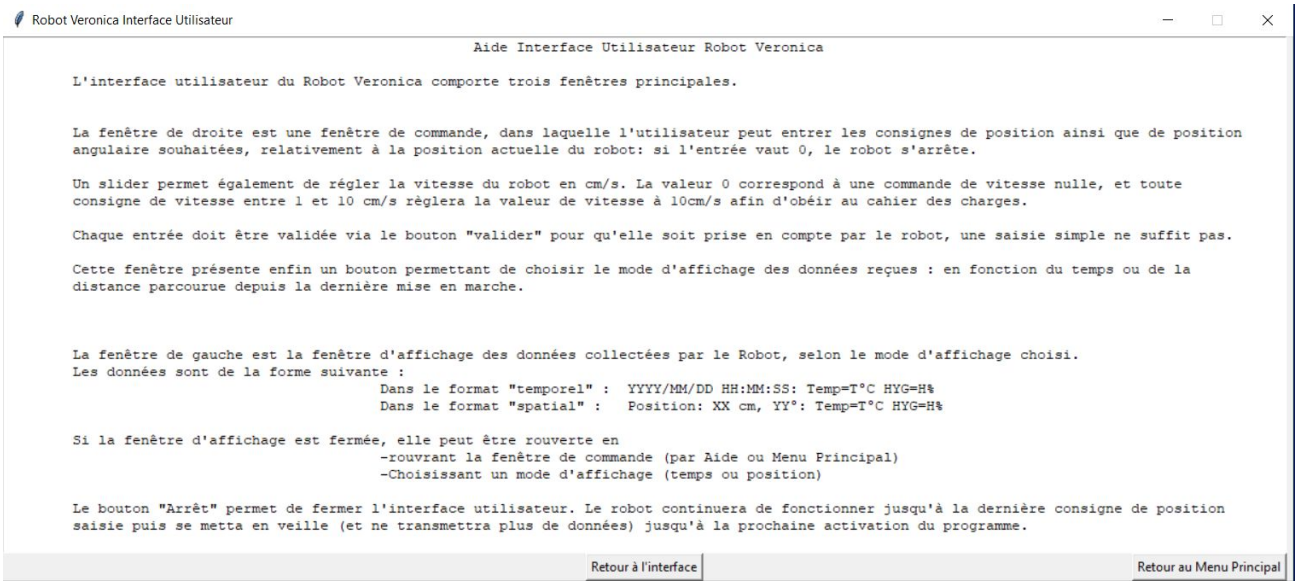


Figure 13: Aperçu de la fenêtre d'aide.

- **Analysis_Page** : Cette classe gère l'affichage dynamique d'un graphique ainsi que des valeurs moyennes sur la session d'utilisation en cours.
  Elle permet de plus de sauvegarder les données dans un fichier texte.

```
656
657         #Page Title
658         Analysis_title = tk.Label(self, text="Page d'Analyse des données",font=('Verdana',20))
659         Analysis_title.place(x=600,y=0,anchor="n")
660
661
662         #strings used to print mean values
663         self.T_str=""
664         self.H_str=""
665
666
667         #Save file entry, give the name of the file data will be saved in
668         self.Save_File=tk.Entry(self,width=15)
669         self.Save_File.place(x=1000,y=600,anchor="e")
670
671
672         #Save file button
673         Save_bt = tk.Button(self, text="Sauvegarder",
674                             command=self.Save_Data)
675         Save_bt.place(x=900,y=600,anchor="e")
676
677
678         #Average Temperature display
679         self.Average_Temperature = tk.Label(self, text="Température Moyenne ={}°C".format(controller.avg_Temp),font=('Verdana',20))
680         self.Average_Temperature.place(x=950,y=300,anchor="n")
681
682
683         #Average Hydrometry display
684         self.Average_Hygrometry = tk.Label(self, text="Hygrométrie Moyenne ={}%".format(controller.avg_Hyg),font=('Verdana',20))
685         self.Average_Hygrometry.place(x=950,y=350,anchor="n")
686
687
688         #Back to interface button
689         Back_Interface = tk.Button(self, text="Retour à l'interface",
690                             command=lambda: controller.show_frame(Main_Page))
691         Back_Interface.place(x=1200,y=0,anchor="ne")
692
693
694         #Help page button
695         Help_bt = tk.Button(self, text="Aide",
696                             command=lambda: controller.show_frame(Main_Page))
697         Help_bt.place(x=1090,y=0,anchor="ne")
698
699
700         #Graph to be displayed on the page
701         self.plot_graph()
```

Figure 14: Attributs principaux de la classe Analysis_Frame

Parmi les attributs définissant la classe, on trouve les attributs suivants :

-**T_str** et **H_str** : chaînes de caractères vides permettant de stocker l'affichage des valeurs moyennes de température et d'hygrométrie.

-**Save_File** : Zone d'entrée de texte permettant à l'utilisateur de spécifier le nom du fichier dans lequel il souhaite enregistrer les données.

-**Average_Temperature** et **Average_Hygrometry** : labels dans lesquels vont être affichées les valeurs moyennes de température et d'hygrométrie.

-**plot_graph** : appel à la méthode **plot_graph** détaillée ci-dessous permettant de créer et placer la zone correspondant au graphique.

On détaille ci-dessous les méthodes inhérentes à cette classe :

-**plot_graph** : permet de créer le Canvas dans lequel le graphique (objet **matplotlib.figure.Figure**) sera inséré.

Cette méthode permet également d'insérer la barre des tâches par défaut matplotlib permettant, entre autres, de sauvegarder le graphique.

-**Update_avg** : permet de mettre à jour les attributs **T_str** et **H_str**, le fonctionnement est similaire à celui de la méthode **Display_Data**.

-**Save_Data** : permet d'enregistrer les données dans un fichier texte dont le nom est défini par l'utilisateur au moyen de l'attribut **Save_File**, correspondant au chemin local (si juste un nom de fichier est entré par l'utilisateur, le fichier sera enregistré dans le répertoire dans lequel le code est stocké).

Si un nom de fichier (ou chemin à partir de l'emplacement du code) est spécifié, alors on définit le chemin absolu comme le répertoire de travail (emplacement du code à partir de la racine) auquel on associe le nom du fichier (ou chemin relatif).

On ouvre ensuite le fichier avant de lui ajouter les données, le créant s'il n'existe pas encore, puis on le referme.

Enfin, on réinitialise l'entrée **Save_File** afin de pouvoir entrer un nouveau nom de fichier si nécessaire.

Le code correspondant à la méthode est donné ci-dessous :

```
743
744     def Save_Data(self):
745
746         '''Saves data in text file
747            Inputs: self
748            Outputs: None
749            Author: FRESLIER Clément, 29/04/2020                    '''
750
751         #Relative path in which the file is
752         rel_path = str(self.Save_File.get())
753
754         #If a file name has been entered, set the absolute path using the SCRIPT_DIT constant
755         if self.Save_File.get()!="":
756             filename=os.path.join(SCRIPT_DIR, rel_path)
757
758             #saving data in the text file
759             savefile=open(filename,"a")
760             for Position, Angle, Date, Temperature, Hygrometry in self.controller.data:
761                 savefile.write("Position= {} cm , Angle= {} ° , Date= {} , Température= {} °C, Hygrométrie = {} %
\n".format(Position,Angle,Date,Temperature,Hygrometry))
762
763             #closing the text file
764             savefile.close()
765
766             #Resetting the filename entry
767             self.Save_File.delete(0,"end")
768             self.Save_File.insert(0, "")
769             filename=""
770
```

Figure 15: Code de la méthode Save_Data

-**Animate** : permet d'animer le graphique de la page d'analyse des données.

Cette méthode récupère les données de **data** et les convertit en objets de type **int** ou **datetime** afin d'afficher la température et le taux d'humidité en fonction de la date d'acquisition (au format HH :MM :SS).

Ces valeurs sont, si la liste de donnée n'est pas vide, ajoutées sur un graphique possédant un axe des abscisses défilant et deux axes des ordonnées.

On ajoute enfin le titre des axes et du graphique.

Le code correspondant est donné ci-dessous.

```python
    def animate(self,i):

        '''refreshes graph values
        Inputs=self, i (interval, not used in the function but in the animation method)
        Outputs: None
        Author: FRESLIER Clément, 28/04/2020                                    '''

        #Variables will be stores in the corresponding lists
        Pos_List=[]
        Ang_List=[]
        Date_List=[]
        Temp_List=[]
        Hyg_List=[]

        #Storing position and formatting them to float or Datetime objects
        for Position_str, Angle_str, Date_str, Temperature_str, Hygrometry_str in self.controller.data:

            Position=float(Position_str)
            Angle=float(Angle_str)
            Temperature=float(Temperature_str)
            Hygrometry=float(Hygrometry_str)

            Date=datetime.strptime(Date_str, "%Y/%m/%d %H:%M:%S")
            Date.strftime("%H:%M:%S")

            Date_List.append(Date)
            Pos_List.append(Position)
            Ang_List.append(Angle)
            Temp_List.append(Temperature)
            Hyg_List.append(Hygrometry)

        ax1.clear()
        ax2.clear()
        delta_t=timedelta(seconds=20)

        #if there's something to plot, plot it as a function of the date with moving x axis
        if len(Date_List)!=0:
            ax1.plot_date(Date_List,Temp_List,'r-',label="Température")
            ax1.legend(loc='upper right')
            ax1.set_xlim(left=max(Date_List[0],Date_List[self.controller.x_index]-delta_t),right=(Date_List[-1]+timedelta(seconds=5)))

            #Secondary y axis
            ax2.plot_date(Date_List,Hyg_List,'g:',label="Hygrométrie")
            ax2.legend(loc="upper right",bbox_to_anchor=(1,0.95))
            self.controller.x_index+=1

        #Labels and graph Title
        ax1.set_xlabel('Temps')
        ax1.set_ylabel('Température (°C)')
        ax2.set_ylabel('Hygrométrie (%)')
```

Figure 16: Code de la méthode d'animation du graphique

Il est possible, à partir de cette page, d'ouvrir la page d'aide ou de retourner au panneau de commande.

Un aperçu de la fenêtre est donné ci-dessous :

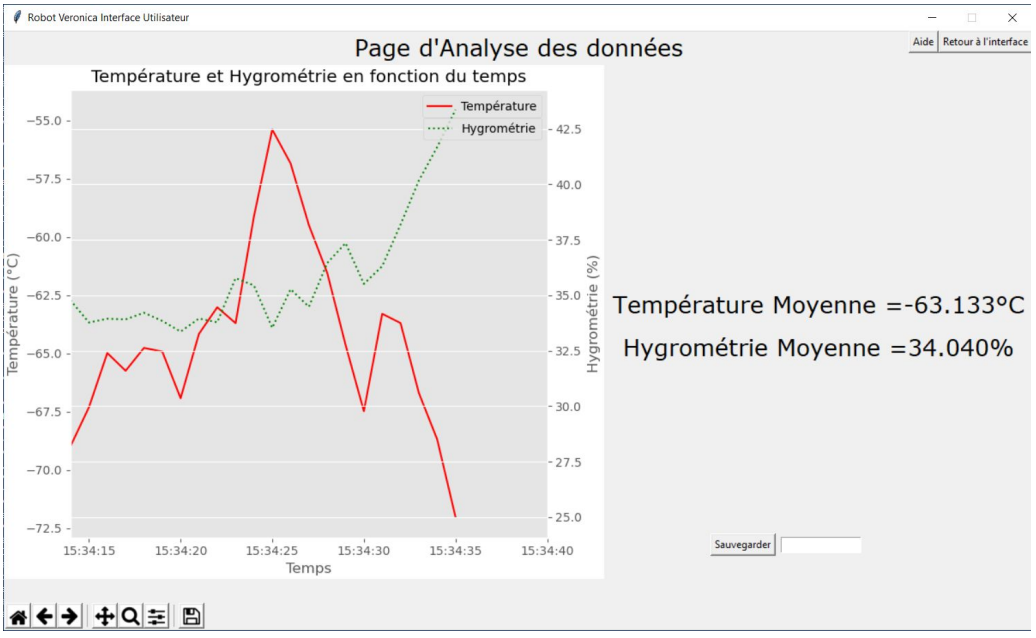


Figure 17: Aperçu de la fenêtre d'analyse. Les données ne correspondent pas aux valeurs affichées sur la figure 12

- **Retrieve_Data** : fonction externe à toute classe permettant de récupérer les données transmises par le robot en temps réel dans un thread second thread.

Cette fonction récupère, lorsqu'il y en a, les données envoyées par le robot (liste **New_Data** passée en argument de la fonction, terminant par **True** si le robot a envoyé des données) et ajoute ces données à la liste **data**.
Le programme calcule ensuite les valeurs moyennes qu'il stocke dans les attributs **avg_Temp** et **avg_Hyg** de la classe **HMI_Veronica**.
La fonction appelle ensuite la méthode **Update** de la classe **Display_Page** afin de mettre à jour l'affichage des données.

```
831
832 '''Function surveying if new data has arrived and, once it has, stores it in the data attribute from HMI_Veronica class
833 ------------------------------------------------------------------------------'''
834 def Retrieve_Data(New_Data,HMI_Veronica):
835
836     '''Surveys in real time if new data has arrived and if so, adds it to 'data'
837        Inputs: New_Data ->list of new data, format [Position1, Angle1,Date1,Temperature1,Hyrometry1,Position2,....,True]
838                HMI_Veronica ->Main class of the application
839        Outputs: None
840        Author: FRESLIER Clément, 26/04/2020                                  '''
841
842     AvgT=[]
843     AvgH=[]
844
845     while True:
846
847         #Sets the function to sleep so that it doesn't take too much resources
848         sleep(0.01)
849
850         #When new data is received
851         if New_Data[-1]==True:
852
853             New_Data.pop()
854
855             #While the list isn't empy, stores values in relevant variables
856             while New_Data:
857
858                 Hygrometry=New_Data.pop()
859                 Temperature=New_Data.pop()
860                 Date=New_Data.pop()
861                 Angle=New_Data.pop()
862                 Position=New_Data.pop()
863
864
865                 #stores those variables in the data list
866                 HMI_Veronica.data.append([Position,Angle,Date,Temperature,Hygrometry])
867
868                 AvgT.append(float(Temperature))
869                 AvgH.append(float(Hygrometry))
870
871             HMI_Veronica.avg_Temp=sum(AvgT)/len(AvgT)
872             HMI_Veronica.avg_Hyg=sum(AvgH)/len(AvgH)
873
874             #if the Display Page exists, call Update method
875             if  HMI_Veronica.frames[Main_Page].Data_Page:
876
877                 HMI_Veronica.frames[Main_Page].Data_Page.Update()
878
879             #Add False to the new data list to specify that no new data is there to display, until new data is received
880             New_Data.append(False)
881
```

Figure 18: Code de la fonction de récupération des données

4. Difficultés techniques

De nombreuses difficultés ont été rencontrées lors du développement de l'application. Les difficultés principales ainsi que les méthodes trouvées pour pallier ces difficultés sont données dans cette section.

- **Gestion des pages** :
  La page d'affichage des données héritant de la classe **tk.Toplevel**, celle-ci restait en permanence au premier plan, même lorsque le menu principal et la page d'aide étaient appelées au premier plan. Ces pages apparaissaient donc derrière la page d'affichage.
  Afin de résoudre de problème, la solution considérée était de forcer, à chaque appel d'une page autre que le panneau de commande, la page d'affichage à l'arrière-plan, et la forcer au premier plan lorsque le panneau de commande était mis au premier plan.
  Les commandes suivantes ont permis d'accomplir cela :

```
164             #Sending Display Page to the background if it exists
165             if self.frames[Main_Page].Data_Page:
166                 self.frames[Main_Page].Data_Page.withdraw()
167
186             #if a Display Page already exists but is sent to the background
187             elif self.frames[cont].Data_Page:
188
189
190                 self.frames[cont].Data_Page.deiconify()
191
```

Figure 19: commandes permettant d'envoyer la fenêtre d'affichage des données en arrière-plan (lignes 165-166) et au premier plan (lignes 187-190)

- **Gestion des données en temps réel** :
  Afin de pouvoir afficher les données en temps réel, il était important de pouvoir traiter celles-ci en temps réel.
  La solution considérée pour résoudre ceci était d'effectuer la récupération et le traitement des données dans un thread différent à l'aide de la programmation parallèle. Pour cela on a importé la fonction **Thread** du module **threading**, et qui permet d'effectuer un programme dans un thread différent.
  Ce thread est initialisé dans le fichier de test qui sera présenté dans la section suivante.

- **Gestion de l'affichage graphique en temps réel** :
  La gestion des animations du graphique était également un problème assez complexe, qui a été résolu en définissant une fonction d'animation du graphique basée sur celle donnée dans la référence [4] (Voir bibliographie argumentée, section Interface Homme Machine)

5. Fichier de test

Dû aux conditions de travail en confinement, et aux contraintes supplémentaires liées, il n'a pas été possible de tester le programme en conditions réelles d'utilisation.

Afin de pouvoir tester le bon fonctionnement de l'interface, un programme de test **Run_file.py** a été réalisé.

Ce programme permet de simuler l'arrivée de données fournies par le robot, qui sont ensuite traitées comme telles.

Ces données simulées sont de la forme suivante :

- Une valeur de position correspondant à une progression linéaire du robot (en cm), à laquelle on ajoute une perturbation aléatoire d'amplitude égale à l'erreur de position maximale admissible.
- Une valeur d'angle correspondant à une rotation linéaire du robot à laquelle on ajoute une perturbation aléatoire d'amplitude égale à l'erreur de position angulaire maximale admissible.
- La date à laquelle les mesures sont prises
- La valeur de température, initialement fixée à -63°C, correspondant à la température moyenne à la surface de Mars (source : Wikipédia), à laquelle on ajoute une perturbation aléatoire correspondant aux fluctuations de température à la surface en fonction du temps et de la position.
- La valeur du degré d'hygrométrie, initialement fixée à 25% de manière arbitraire, la valeur moyenne n'ayant pas été trouvée, à laquelle on ajoute une perturbation aléatoire correspondant aux fluctuations du taux d'humidité à la surface en fonction de temps et de la position.

Ces données sont placées dans une fonction **wait** permettant d'actualiser ces données toutes les secondes grâce à la commande **sleep(1)** du module **time.**

Ce programme tourne lui aussi dans un thread à part afin de pouvoir générer les données en temps réel, et les traiter de même.

Les threads sont initialisés par le commande suivante :

```
68
69 #main thread (uses Retrieve_Data function from Robot_Veronica.py
70 t=Thread(target=Retrieve_Data,args=(Buffer_List,robot),daemon=True)
71 t.start()
72
```

Figure 20: création du thread de récupération et traitement des données

Le paramètre **daemon=True** permet quant à lui de fermer le thread à l'arrêt du programme.

On lance également la fonction d'animation avec le module **FuncAnimation** de la librairie **matplotlib**, avec un interval de 1 seconde (**interval=1000**)

```
65
66 #animation function
67 anim=animation.FuncAnimation(fig,robot.frames[Analysis_Page].animate,interval=1000)
```

Figure 21: Lancement de l'animation du graphique

6. Notes

Cette partie présente les différents détails qu'il a fallu prendre en compte pour rendre l'interface la plus ergonomique possible, ainsi que des notes utiles pour le bon fonctionnement du programme.

- **Détails à prendre en compte lors de la création de l'IHM :**
  - Afin de s'assurer que l'utilisateur ne ferme pas le programme par erreur, il a été nécessaire de définir la méthode **Callback** permettant d'empêcher la fermeture systématique de l'interface à l'aide du Windows Manager.
  - Afin d'avoir un affichage ergonomique, le placement des fenêtres s'effectue en fonction de la taille de l'écran de l'utilisateur, les commandes correspondantes sont données sur la figure ci-dessous

```
142
143         #screen size
144         w,h=self.winfo_screenwidth(),self.winfo_screenheight()
145         x,y=int(w/2-230),int(h/2-155)
146
147         #Positionning the page at the center of the screen with given size
148         self.geometry('460x310+'+str(x)+'+'+str(y))
149
```

Figure 22: commandes permettant de placer la fenêtre du menu principal au milieu de l'écran de l'utilisateur

  - Afin d'éviter les erreurs de chemin lors de l'importation des images ou de l'enregistrement des fichiers, le chemin doit être récupéré automatiquement par le programme et non rentré manuellement dans le cas d'un chemin absolu d'où l'utilisation de la commande suivante :

```
57 ----------------------------------------------------------------------
58 SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
```

Figure 23: commande récupérant le chemin d'accès au programme à partir de la racine

  - La fenêtre d'affichage des données devait pouvoir être fermée lorsque l'utilisateur souhaitait changer de page. Cet aspect a été traité dans la partie consacrée aux difficultés rencontrées.

- **Notes :**
  - Si le programme affiche l'erreur suivante :

```
PS C:\Users\Clément F> py .\Test_graph.py
D:\Python_3_8_2\python.exe: can't open file '.\Test_graph.py': [Errno 2] No such file or directory
PS C:\Users\Clément F>
```

Il faut alors vérifier le répertoire de travail avec la commande **os.getcwd()** du module os et le changer si nécessaire.

- Afin de changer le répertoire de travail, on peut utiliser dans l'interpréteur Python la commande **os.chdir(r 'paths')** où 'paths' est le répertoire de travail (absolu ou relatif)
- Ne pas exécuter sous Pyzo. Préférer l'interpréteur Python Shell ou Windows PowerShell
- Tous les tests effectués sur cette interface ont été réalisés avec le Windows PowerShell avec les commandes suivantes :

```
PS C:\Users\Clément F> cd "D:\Users\Clément F\Documents\PROTIS\HMI_Veronica"
PS D:\Users\Clément F\Documents\PROTIS\HMI_Veronica> py .\Run_file.py
```

III. Communication

La communication entre le robot et un ordinateur est un aspect essentiel de la mission. Il faut s'assurer que la communication soit fiable et sécurisée. Le moyen de télécommunication qui sera utilisé ici sera le Bluetooth et le programme sera rédigé en Python. Afin de comprendre le fonctionnement du programme nous allons revenir sur quelques points concernant Python et le Bluetooth.

1. Présentation générale

A. Bluetooth

Bluetooth est une technologie de liaison sans fil, utilisant les ondes radio, sur une fréquence comprise entre 2.4 et 2.483Ghz. Elle permet de relier tout type de périphérique comprenant une puce Bluetooth, que ce soient des imprimantes, des modems, des disques durs externes, des congélateurs et même des télévisions...

Le protocole de gestion de liaison est responsable de l'établissement de la connexion entre les éléments Bluetooth. Cette couche du protocole se charge de fournir des outils de sécurité comme l'authentification, le cryptage par génération, échange et vérification des clés de liaison et de cryptage. Il gère aussi le contrôle et la négociation de la taille des paquets de bande de base.

Le protocole RFCOMM est la base du remplacement des fils par Bluetooth. C'est un simple protocole de transport, il supporte une grande quantité d'application qui utilisent la communication par ports série. RFCOMM fournit un transfert de donnée fiable, de multiples connexions simultanées, le control de flux et les caractéristiques d'une ligne série.

Un appareil peut proposer de nombreux services Bluetooth, chacun de ces services doit utiliser un protocole donné. Nous allons nous concentrer sur les services utilisant le protocole RFCOMM. Chaque service proposé par un appareil utilise un port donné pour se connecter à d'autres appareils.

Bluetooth permet la constitution de réseaux qui communiquent par ondes radio dans un rayon de 10 mètres, actuellement, et jusqu'à 100 mètres dans un avenir proche. On nomme les réseaux ainsi constitués des piconets (**figure 24**).

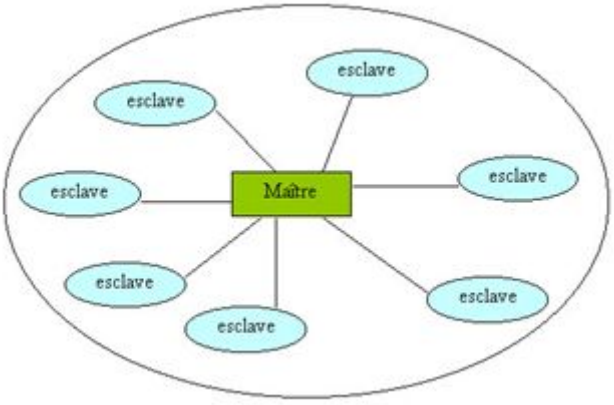


Figure 24 *: Architecture d'un piconet*

B. Python

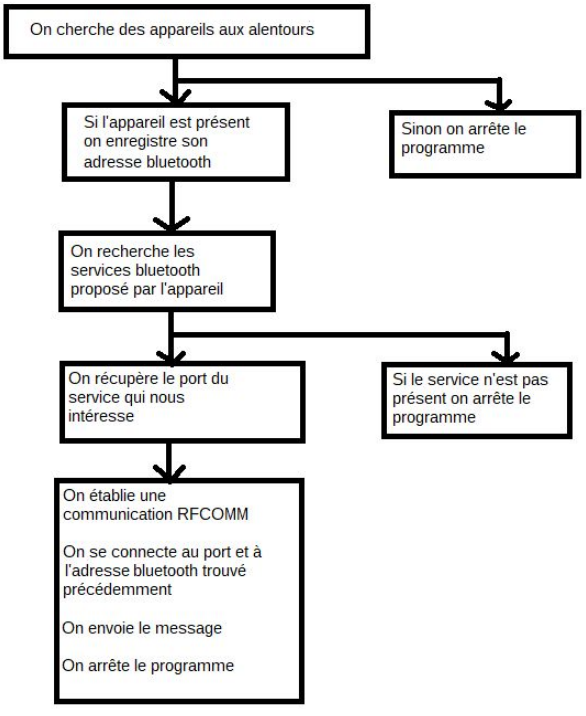
Python est un langage de programmation open-source orienté objet qui utilise une syntaxe claire ainsi qu'une gestion de la mémoire intégré dans le langage. Cela a l'avantage de laisser l'utilisateur se concentrer sur la programmation sans se soucier de la gestion de la mémoire. De plus Python possède une vaste librairie elle aussi open-source. La gestion simple du Bluetooth ne fait pas partie du langage standard.

Nous allons donc utiliser le module PyBluez qui fournit un accès aux ressources Bluetooth dans un langage orienté objet.

2. Architecture de la communication

Le module PyBluez bien que très complet ne permet pas d'échanger des informations de manière simultané. Dès qu'un message est envoyé il est nécessaire que l'appareil qui émette soit le « maître » et que l'autre appareil soit « l'esclave ». On va donc devoir écrire un programme qui gérera l'envoie et un programme qui gérera la réception de messages.

A. Envoi



L'architecture du programme d'envoi est la suivante (**figure 25**) :

-On recherche les appareils présents aux alentours.

-Si l'un des appareils possède le nom de l'appareil auquel on cherche à se connecter alors on enregistre son adresse Bluetooth.

-On recherche les services proposés par cet appareil

-On récupère le port du service auquel on souhaite se connecter.

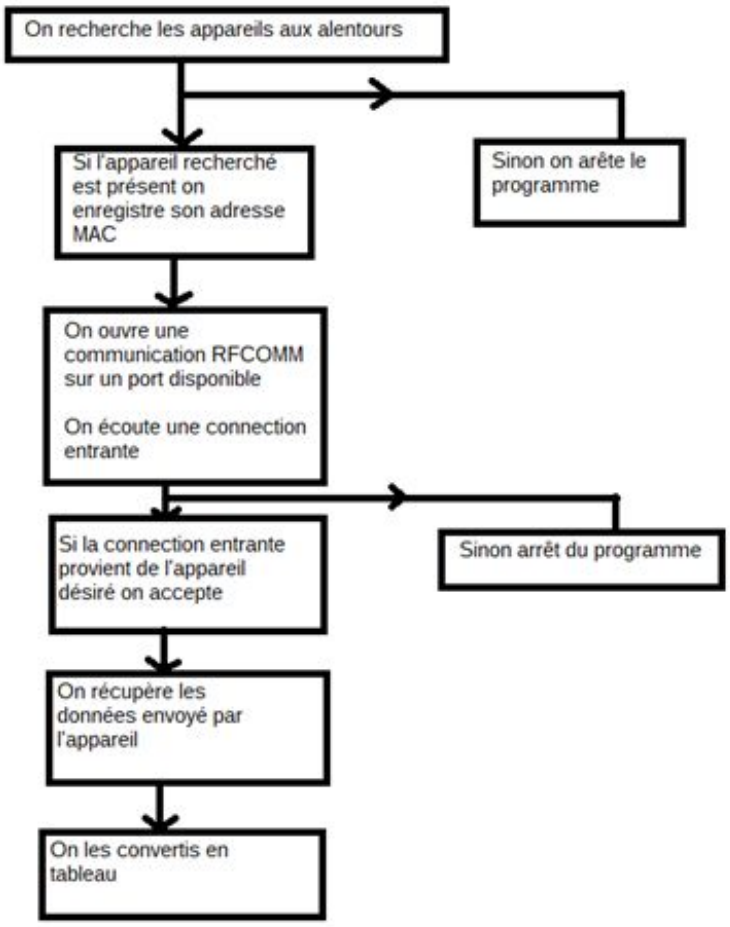
-On établit la communication.

Le détail de chacune de ces fonctions est décrite aux paragraphes II-3-a,II- 3-b et II-3-c.

Figure 25 *: Architecture du programme d'envoi*

B. Réception

L'architecture du programme de réception est la suivante (**figure 26**) :



-On recherche les appareils aux alentours

-Si l'appareil recherché est présent on enregistre son adresse Bluetooth

-On ouvre une communication RFCOMM

-On écoute une connexion entrante

-On vérifie que la connexion entrante provient bien de l'appareil désiré

-On récupère les données

Le détail de chacune de ces fonctions est décrite II-3-A, II-3-D.

***Figure 26** : Architecture du programme de réception*

3. Envoi et réception de messages

A. Choix d'un partenaire de communication

Chacun de ces programmes commence par rechercher les appareils Bluetooth aux alentours. PyBluez représentent les adresse Bluetooth comme une chaîne de caractère sous la forme : « xx:xx:xx:xx:xx », où chaque « x » est un caractère hexadécimal. Choisir un appareil Bluetooth signifie de choisir son adresse Bluetooth.

Si seul le nom de l'appareil est connu, il faut alors scanner les appareils Bluetooth aux alentours à l'aide de la fonction *discover_devices()*. Cette fonction scan l'environnement pendant environ 10 secondes et retourne la liste

des adresses des appareils détectés aux alentours ainsi que la liste des appareils déjà apparié (adresse Bluetooth en vert et nom des appareils en rouge) :

[('DC:2C:26:01:2C:6B', 'JETech 0884'), ('FC:58:FA:D0:02:27', 'PHILIPS BTD2180'), ('2C:41:A1:FE:65:C4', 'Bose SoundSport'), ('00:3D:E8:3E:B7:E6', 'G6'), ('4C:87:5D:82:6D:56', 'Bose QC35 II'), ('70:5E:55:F0:C8:EC', 'realme 3 Pro')]

On peut ainsi récupérer l'adresse de l'appareil auquel on souhaite se connecter.

B – Recherche des services

La recherche des services proposé par un appareil peut être effectué grâce à la fonction *find_service()*. On peut alors avoir accès à tous les services proposés par un appareil donné. Les services peuvent contenir plusieurs informations :

- Name: b'v1.4'
- Description:
- Protocol: RFCOMM
- Port: 4
- Service id: None

Il y a tout d'abord le nom du service, celui-ci est systématiquement présent. Il peut y avoir une description du service. Le protocole utilisé est spécifié ainsi que le port sur lequel il faut se connecter pour accéder à ce service. Enfin on peut y voir un ID, celui-ci correspond à l'UUID associé à ce service. Les UUID sont des identifiants uniques codés sur 128 bits et sont produits en utilisant des composantes pseudo-aléatoires. Il n'est pas nécessaire de connaître l'UUID pour envoyer un message.

Il est nécessaire de connaître au préalable le service auquel on souhaite se connecter afin de pouvoir récupérer le port associé à ce service.

C. Envoi de messages

Une fois l'adresse Bluetooth et le port récupéré il est désormais possible d'établir un socket à l'aide de *BluetoothSocket()*. Un socket représente un point de communication. PyBluez supporte deux protocoles de communication : RFCOMM et L2CAP. Le protocole utilisé doit être mis en argument de la fonction *BluetoothSocket()*. Nous allons utiliser ici le protocole RFCOMM.

Le socket utilisé avec un protocole RFCOMM pour les communication sortante doit se connecter à l'aide de la fonction *connect* où il faut spécifier l'adresse Bluetooth et le port auquel on cherche à se connecter.

L'envoi de messages se fait une fois que la connexion a été établie à l'aide de la fonction *send*. La fonction *send* ne permet d'envoyer que des chaînes de caractère. La partie affichage de ce projet utilise des listes, il est donc nécessaire de convertir les informations une fois reçu.

D. Réception de messages

Afin de recevoir un message, un socket doit être ouvert. On doit ensuite choisir un UUID et un port. Il est nécessaire que le port soit un port qui ne soit pas utilisé par un autre service. L'UUID peut être pris au hasard, il suffit qu'il corresponde au format suivant : « xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx » où chaque « x » est un caractère hexadécimal.

On peut alors ouvrir un socket à l'aide de *BluetoothSocket*. Il faut ensuite écouter les connections entrantes à l'aide de la fonction *listen()*. L'appareil doit également se manifester à l'aide de la fonction *advertise_service()* afin qu'un autre appareil puisse se connecter. Il est nécessaire de spécifier les différentes caractéristiques du service proposé par l'appareil. Il faut ensuite accepter la connexion de l'appareil qui cherche à communiquer avec nous à l'aide de la fonction *accept()*. Enfin on peut recevoir les données à l'aide de la fonction *recv()*.

Enfin comme précisé précédemment, il est nécessaire de convertir les données reçues qui arrivent sous la forme de chaîne de bytes. Cela peut se faire à l'aide du module Python « ast ». On peut utiliser la fonction *decode()* pour récupérer une chaine de caractère puis la fonction *literal_eval()* pour convertir la chaîne de caractère en liste.

E. Sécurité de la communication

La communication est sécurisée simplement. Que cela soit pour les communications entrante ou sortante, on s'assure que la communication s'effectue bien avec l'appareil que l'on souhaite. Si un autre appareil tente de communiquer, la communication est refusé.

4. Résultat

Les programmes de réception nécessite de renseigner le nom de l'appareil duquel on attend un message, si celui-ci envoie un message ce dernier est converti en tableau et affiché (**figure 27**). Le programme d'envoie nécessite le nom de l'appareil auquel on souhaite se connecter, le port du service bluetooth que l'on souhaite utiliser et le message à envoyer. Il renvoie simplement si le message a été envoyé ou pas (**figure 28**).

```
== RESTART: C:\Users\Johan\AppData\Local\Programs\Python\Python38-32\Receive.py =
>>> Receive('G6')
L'adresse de l'appareil G6 est 00:3D:E8:3E:B7:E6
Listening for connection on port:  0
Connecté à  ('00:3D:E8:3E:B7:E6', 3)
[1, 2, 3, 4, 5, True]
>>>
```

Figure 27 : Vue de l'affichage du programme de réception

```
>>> Send('G6',b'v1.4','[1,2,3,4,True]')
L'adresse de l'appareil G6 est 00:3D:E8:3E:B7:E6
Le message [1,2,3,4,True] a été envoyé
>>>
```

Figure 28 : Vue de l'affichage du programme d'envoie

5. Discussion

Comme nous l'avons vu il n'est pas possible d'envoyer et de recevoir des messages de manière simultané à l'aide de PyBluez. Afin de pouvoir échanger des informations dans les deux sens, ils est nécessaire que l'un des appareils exécute le programme d'envoie pendant que l'autre exécute le programme de réception. Il faut que chaque appareil alterne régulièrement entre la réception et la transmission pour pouvoir converser.

Cette partie n'a pas été traité en raison des difficultés supplémentaires lié à la crise du COVID-19. Une étude supplémentaire pourrait être mené pour pouvoir réaliser cette fonction.

IV - Simulation du moteur

1. Présentation générale

Le robot doit pouvoir se déplacer sur la surface de Mars suffisamment rapidement et précisément. Le cahier des charges indique que le robot doit pouvoir se déplacer en ligne droite, selon une distance transmise par l'utilisateur, à une vitesse comprise entre 0,1 et 0,3 m/s, avec une précision de 2 cm. Il doit aussi pouvoir tourner sur lui même d'un angle transmis à distance, avec une précision de 3°. Enfin, il doit pouvoir parcourir 1km sans avoir à recharger ses batteries.

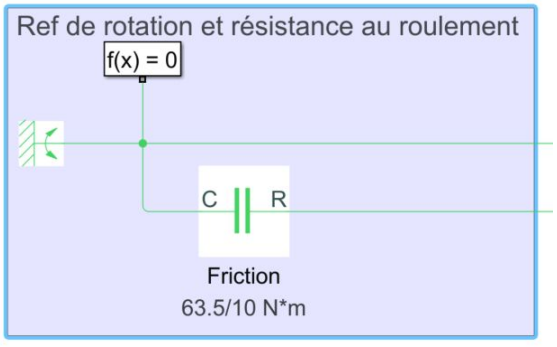
Pour voir si il est possible de produire un robot conforme au cahier des charges, on le simule sur le logiciel Simulink, en utilisant les toolbox *Simscape* et *Simscape Electrical*, qui permettent de simuler la partie mécanique et électrique respectivement.

2. Simulation

On suppose que le robot à deux roues motrices indépendantes, chacune alimentée par un moteur à courant continu. Les moteurs sont contrôlé en *Pulse Width Modulation* (PWM), et leur sens de rotation est contrôlé par un pont en H. Pour simplifier la simulation, on simule un seul moteur qui agit simultanément sur les deux roues. Il faudra donc choisir des moteurs deux fois moins puissant que celui utilisé pour avoir les même performance que la simulation.

a) Simulation de la résistance au roulement

Il faut simuler la résistance au roulement du sol martien. Pour cela on ne prend en compte que la résistance du sable sur les roues, qui induit une force de résistance au roulement de la forme $F = C_{rr}mg$, où m est la masse du véhicule, g est l'accélération de pesanteur locale et C_{rr} est le coefficient de résistance au roulement. On trouve sur Wikipédia que $g_{mars} = 3,771 m/s^2$, que la masse typique d'un rover martien est $m = 185\ kg$, et que pour un pneu dans du sable, $C_{rr} = 0,3$. On utilisera cependant $C_{rr} = 0,03$ car le moteur utilisé n'est pas assez performant. Le couple résistant qui en résulte est $C_r = 6,35\ N.m$. Il n'existe pas de bloc qui permet de simuler la résistance au roulement sans simuler l'intégralité du véhicule et du terrain, ce qui aurait été trop complexe et chronophage. On simule donc ce couple résistant par des frottements induisant un couple nul à vitesse nulle, et un couple C_r sinon.



Le bloc le plus à gauche indique la référence de rotation. Le bloc *Friction* permet de simuler la résistance.

Figure 29 : Bloc de simulation de la résistance au roulement

b) Bloc moteur

Le bloc moteur modélise le réducteur, le moteur, ainsi que le montage électrique qui le contrôle.

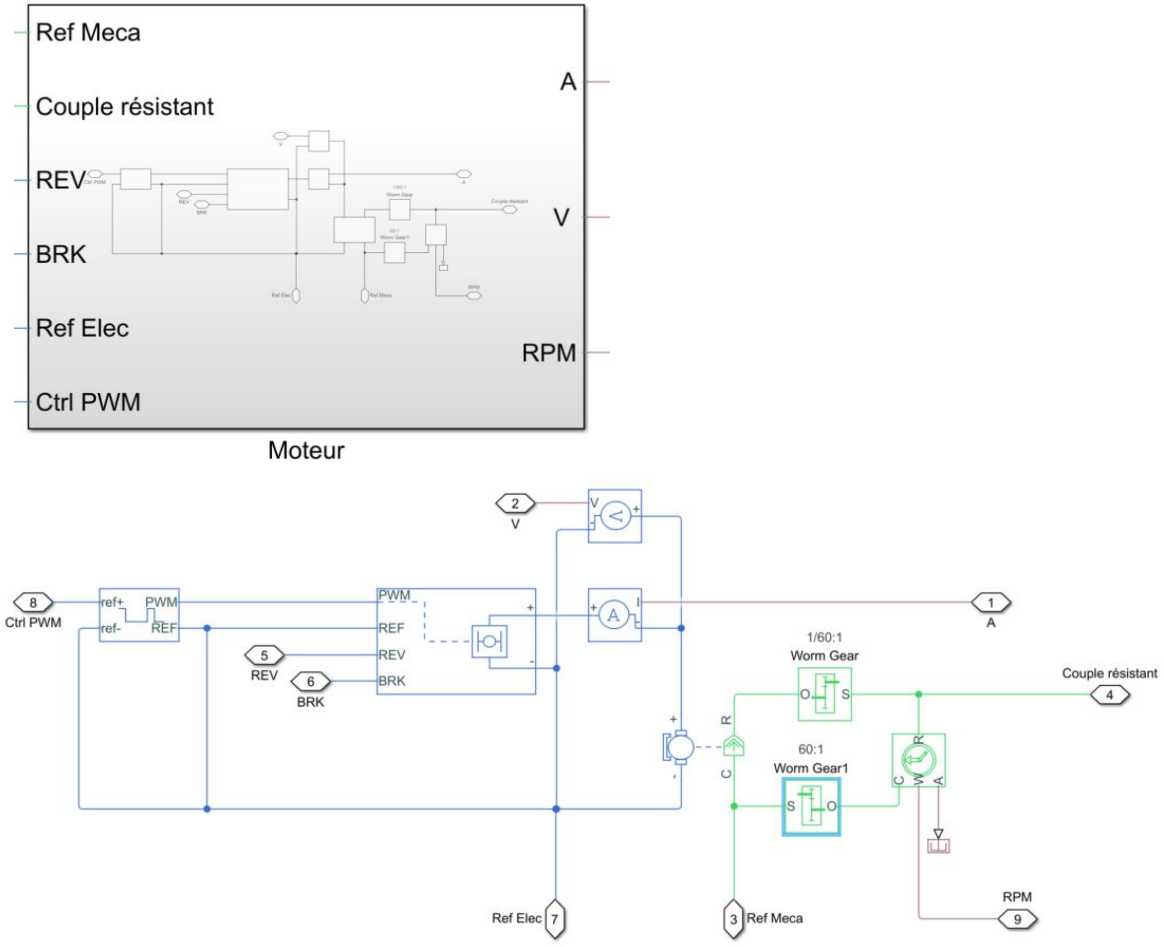


Figure 30 : Bloc moteur

Les blocs qui le compose sont, de gauche à droite :

- Un générateur de signal PWM. La largeur des impulsions sont contrôlées par la différence de tension entre ref+ et ref-. Une différence nul donne un rapport cyclique nul, une différence de 5V donne un rapport cyclique unitaire. Le signal sort de PWM.
- Un pont en H. Il délivre une tension comprise entre -48 et 48 V. Le signe est contrôlé par REV, et l'amplitude par le rapport cyclique du signal PWM.
- Un voltmètre et un ampèremètre, qui mesurent la tension et le courant aux bornes du moteur.
- Le moteur.
- Le réducteur. Celui du haut convertit le couple résistant aux roues en un couple résistant au moteur. Celui du bas convertit la vitesse de rotation du moteur en la vitesse de rotation des roues.
- Un bloc permettant de mesurer la vitesse de rotation au niveau des roues.

Le bloc moteur a pour entrés et sorties :

- Signal électrique d'entré
  - Ref Elec : Modélise la terre
  - Ctrl PWM : Signal compris entre 0 et 5 V, qui contrôle la largeur du PWM

- REV : Signal compris entre 0 et 5 V, il permet d'inverser la polarisation du pont en H lorsqu'il est supérieur à 2,5 V
- BRK : Signal compris entre 0 et 5 V, il permet de court-circuiter le pont en H

- Signal mécanique d'entré
  - Ref Méca : Modélise la référence mécanique
  - Couple résistant : Modélise le couple résistant
- Signal électrique de sortie
  - V : Mesure de la tension au borne du moteur
  - A : Mesure du courant au bornes du moteur
- Signal mécanique de sortie
  - RPM : Mesure de la vitesse de rotation des roues en tours par minute.

Les caractéristique du moteur sont celle du moteur Crouzet 80180504. Ce moteur a été retenue car il le plus performant que nous ayons trouvé sur internet, il ne l'est cependant pas assez (voir partie 'amélioration')

c) Asservissement

Il faut maintenant pouvoir contrôler ce moteur pour piloter le robot. Il faut que le robot soit capable d'accélérer, puis rouler à vitesse constante et enfin décélérer jusqu'à son point d'arriver, en ne connaissant que la distance à parcourir. On remarque lors des simulation que le robot est à la fois rapide et précis lorsqu'il commence à ralentir 1 m avant d'arriver à destination, on choisit donc cette distance pour démarrer la phase de décélération.

On mesure la vitesse de rotation des roues, que l'on convertit en vitesse de déplacement à l'aide de la formule $V = R\Omega$. V est la vitesse du robot en m/s, Ω la vitesse de rotation des roues en rad/s, et R le rayon des roues en m. On prend $R = 0,3\ m$, ce qui correspond à la taille typique d'un rover martien. En intégrant la vitesse, on peut calculer la différence entre la distance parcouru et la distance total, puis calculer une commande pour le moteur.

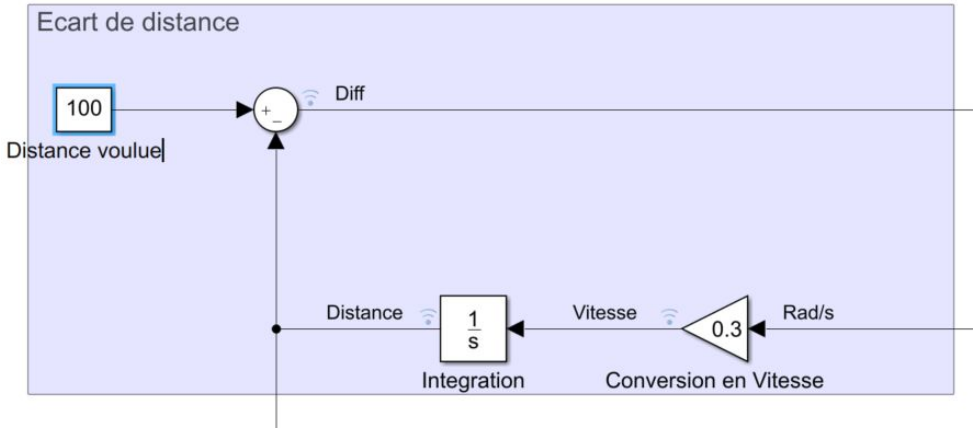


Figure 31 : Calcul de la différence de distance

Si la différence est plus grande que 1 m (respectivement plus petite que -1 m si le robot doit reculer), la commande du moteur vaut 5 V et REV 0 V (resp. REV = 5 V) pour que le robot se déplace à vitesse constante.
Si la différence est comprise entre 1 et -1 m, le robot rentre dans la phase de décélération, et la commande du moteur est calculée par un PID. Il faut faire attention à ce que l'entré du PID soit nulle avant d'entrer dans cette phase, pour ne pas saturer l'intégrateur, et ne pas oublier d'empêcher au PID de sortir des valeur supérieur à 5 (resp. inférieur à -5), car le générateur de signaux PWM n'accepte pas d'entrer supérieur à 5 V.

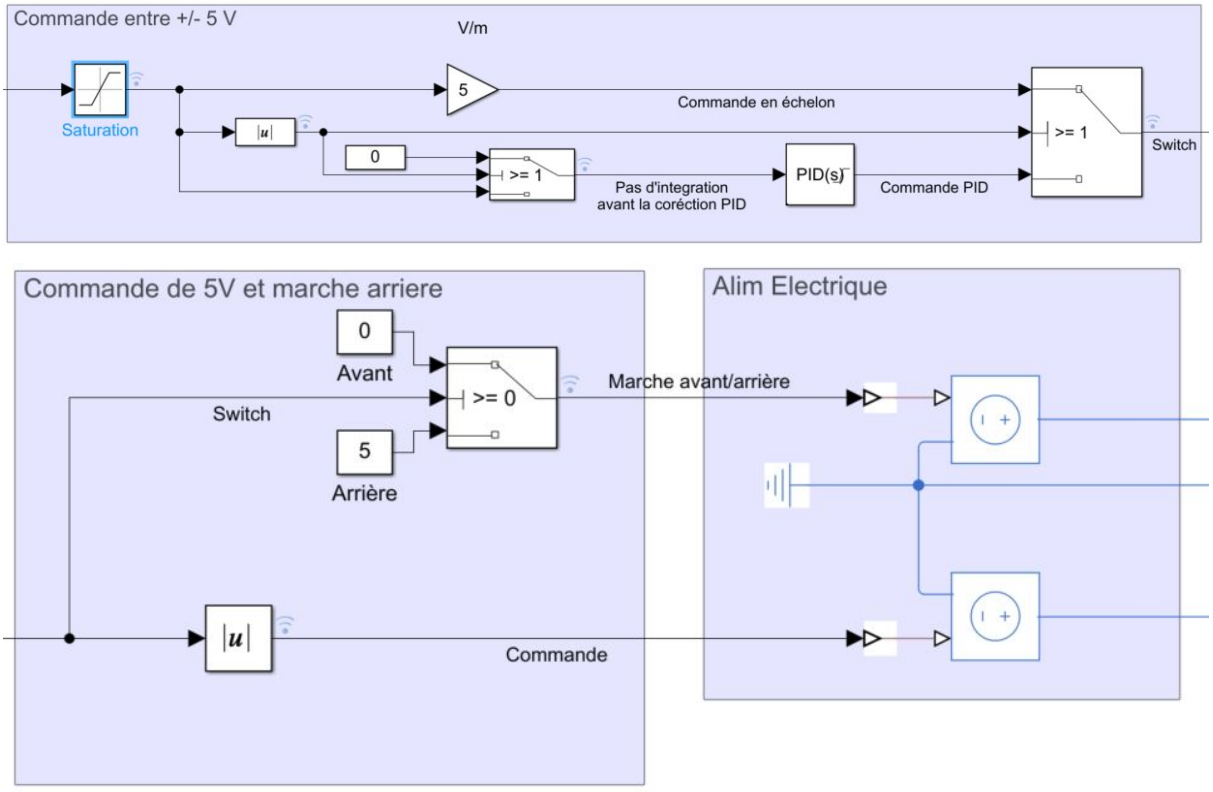


Figure 32 : Calcul de la commande

La différence de distance entre dans le bloc 'Saturation'. La sortie de ce bloc vaut 1 (respectivement -1) si son entrée est supérieur à 1 (resp. inférieur à -1). La sortie est égale à l'entrée sinon. Si la valeur absolue de la différence de distance saturée est égale à 1 m, les deux *switchs* du bloc 'Commande entre +/- 5 V' sont en position haute. Dans ce cas, l'entrée du PID et nulle, et la sortie du du bloc 'Commande entre +/- 5 V' vaut +/- 5.
Si la valeur absolue est inférieur à 1, les *switchs* passent en position basse. L'entrée du PID est alors la différence de distance, et la sortie de 'Commande entre +/- 5 V' est la commande calculée par le PID.

La sortie du bloc 'Commande entre +/- 5 V' rentre ensuite dans le bloc 'Commande de 5 V et marche arrière'. Si la commande d'entré est positive, le *switch* est en position haute, et sa sortie vaut 0. Dans le cas contraire, le *switch* change d'état, et sa sortie vaut 5. La commande de 'Commande entre +/- 5 V' passe ensuite par le bloc 'valeur absolue' pour s'assurer qu'elle reste positive.
Les deux blocs de 'Alim Electrique' permettent de générer une tension égale à la valeur numérique de leur signal d'entrée (au niveau de la flèche blanche).
Le générateur du haut contrôle l'entrée REV du moteur, et le générateur du bas contrôle l'entrée Ctrl PWM.

On a ainsi un signal REV qui vaut 0 V lorsque le robot doit avancer, et 5 V lorsqu'il doit reculer, et un signal Ctrl PWM qui vaut 5 V lorsque le robot est à plus de 1 m de son point d'arrivée, et qui corrige le moteur à l'aide d'un PID lorsque le robot doit s'arrêter.

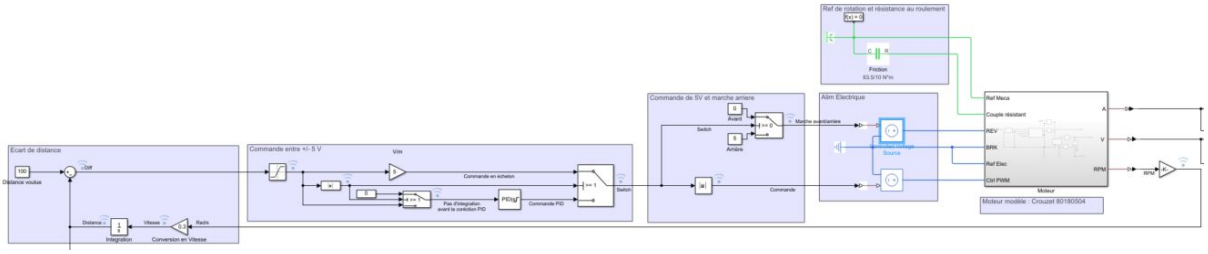


Figure 33 : Schéma bloc complet de la simulation

Dans le cas de la rotation, on suppose que l'angle θ donné par l'utilisateur est donné en degré, dans le sens direct, et on suppose que les deux roues motrices sont séparées de 1 m. Dans ce cas, la roue de droite devra parcourir une distance de $D = \frac{1}{2}\frac{\pi}{180}\theta$ m, et la roue de gauche une distance $-D$. Pour simuler la rotation, il faut simuler chaque roue séparément. L'asservissement ne change pas, mais il faut utiliser un moteur deux fois moins puissant, un couple résistant deux fois plus petit pour chaque roues.

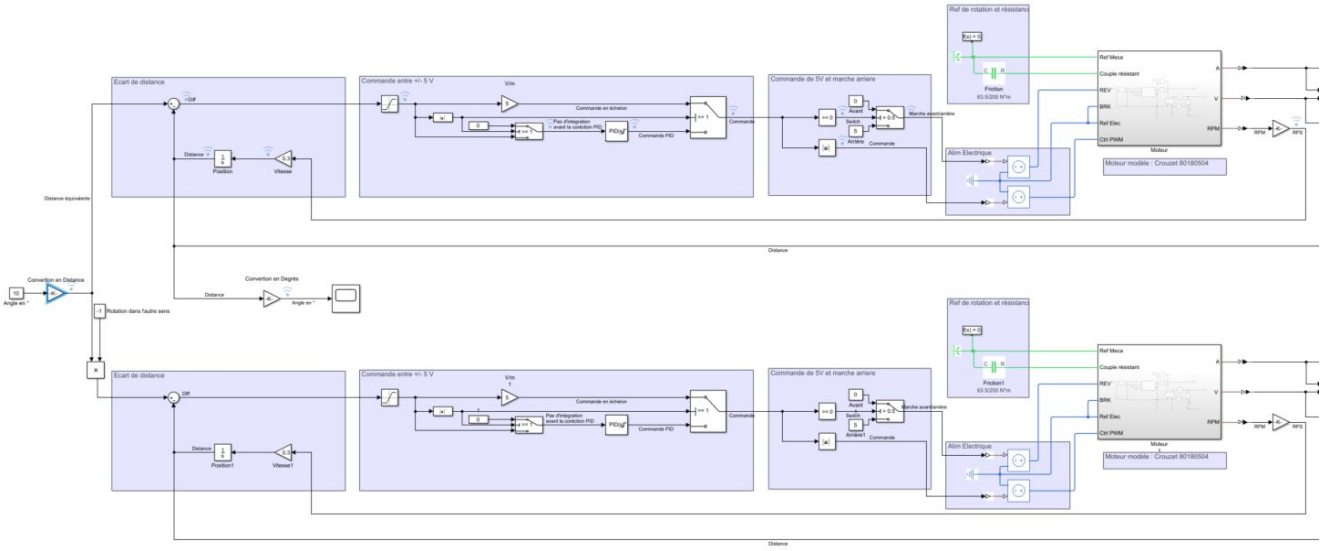


Figure 34 : Schéma bloc complet de la simulation de la rotation

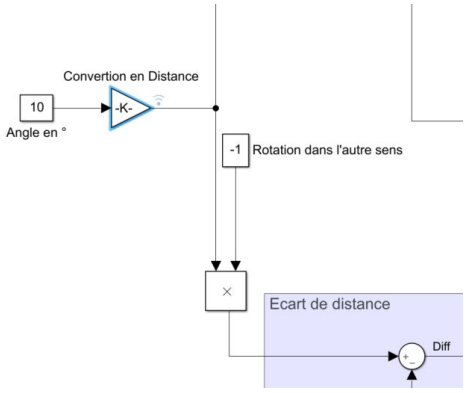


Figure 35 : Conversion en distance

d) Importance du PID

On remarque sur la figure 36 que l'absence du correcteur PID induit un écart de l'ordre de 30 cm, alors que figure 37 montre que la présence d'un PID rend la décélération moins brusque, et que l'écart statique est de l'ordre du millimètre.

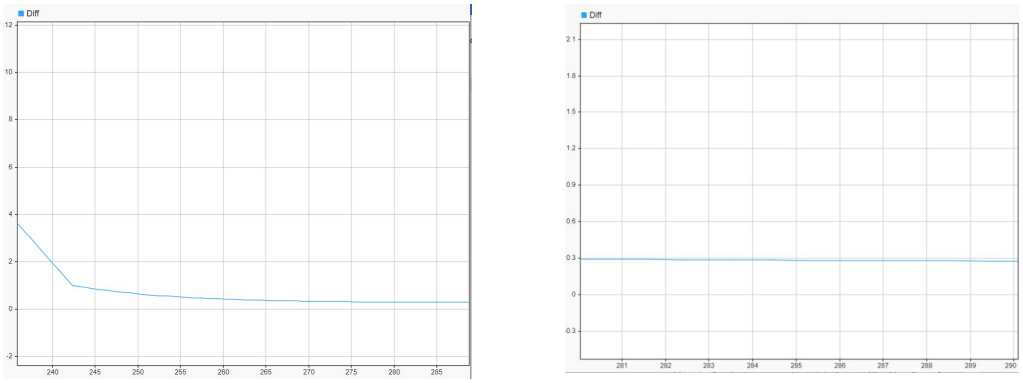


Figure 36 : Différence (m) en fonction du temps (s) sans PID

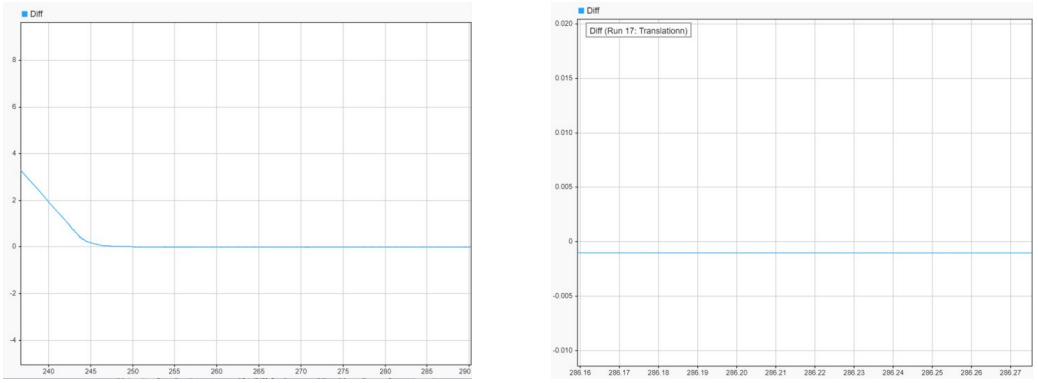


Figure 37 : Différence (m) en fonction du temps (s) avec PID

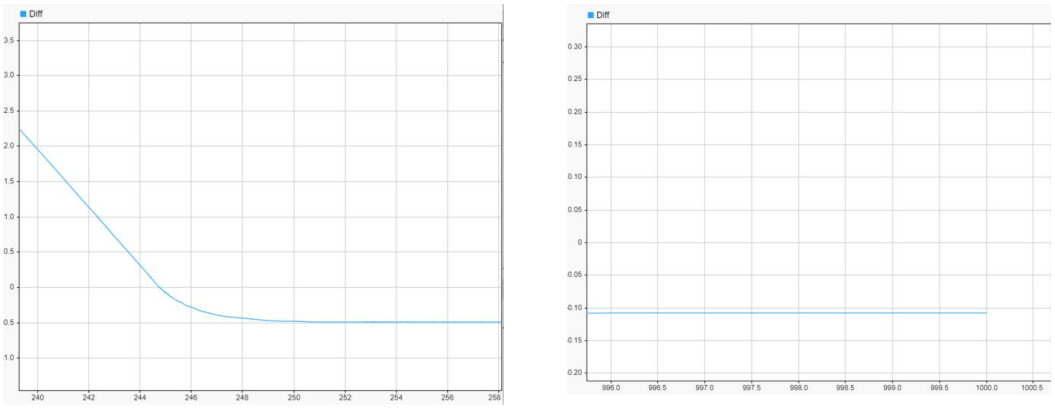


Figure 38 : Différence (m) en fonction du temps (s) avec PID, sans contrôler son entrée

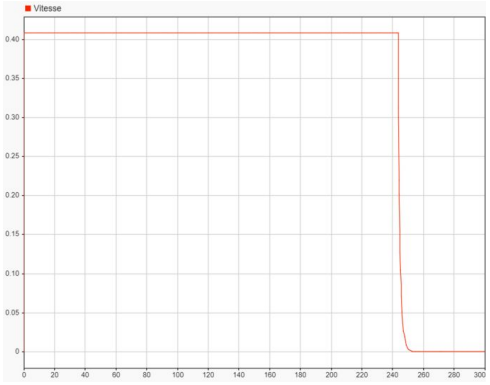
On remarque enfin sur la figure 38 que lorsqu'on oublie de rendre l'entrée du PID nulle avant la phase de décélération, non seulement le PID régit trop tard et le robot dépasse le point d'arrivé de 0,5 m. Après avoir attendue 750 s, l'écart est toujours de 0,1 m. Il est donc essentiel de s'assurer que l'entrée du PID reste nulle jusqu'au début du ralentissement du robot.

Le PID est idéal, de la forme $PID(s) = P(1 + I\frac{1}{s} + D\frac{N}{1+N\frac{1}{s}})$. les valeurs de P, I, D et N, sont calculée avec l'outil intégré de Simulink pour optimiser le temps de d'arret et la précision, pour une distance d'arrêt de 1 m, avec une vitesse initial de 0,4 m/s, qui est la vitesse maximale du robot. $P = 52,02\ V/m$, $I = 2,057e-3\ s^{-1}$, $D = 9,678e-1\ s$ et $N = 3$.

e) Performance

Les performance du robot sont globalement conforme au cahier des charges :

- Se déplace en ligne droite
- Tourne sur lui même
- Vitesse maximale de 0,4 m/s
- Écart de distance inférieur à 2 cm
- Écart de rotation inférieur à 3°
- Autonomie d'une heure



Bien que la vitesse soit trop grande, il est possible de la baisser multipliant la commande par un gain inférieur à 1

Figure 39 : Vitesse (m/s) en fonction du temps (s)

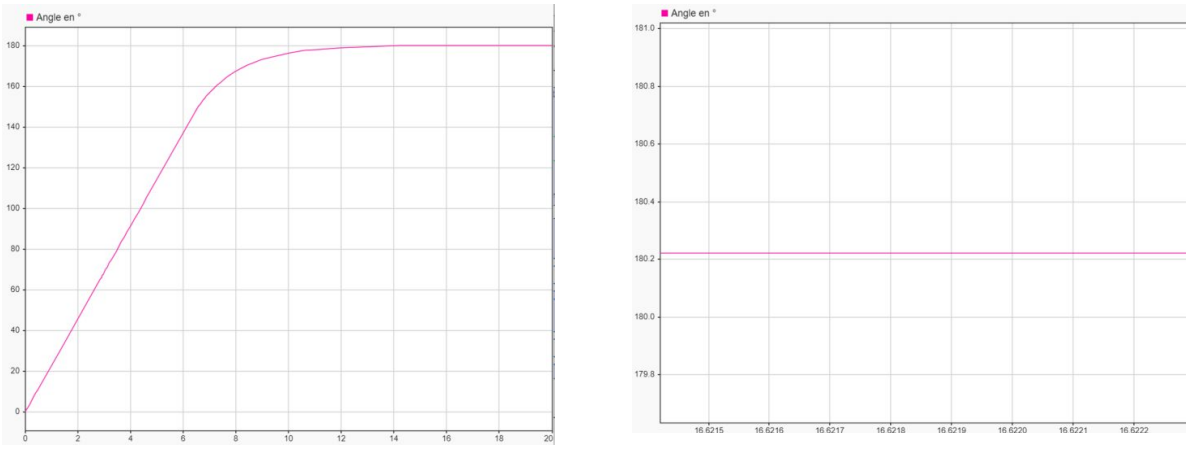
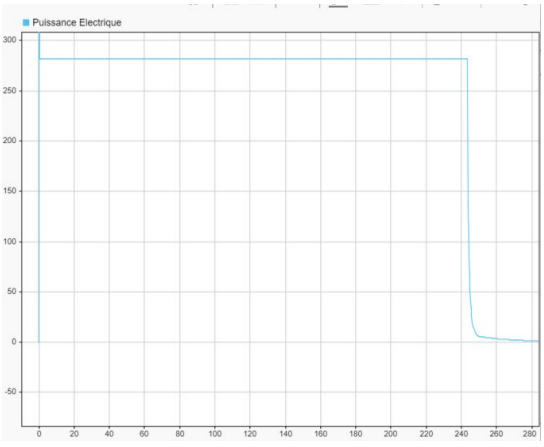


Figure 40 : Angle (°) en fonction du temps (s)

Le robot peut faire un tour sur lui même en 12 seconde, avec un précision de 0,2°.



A sa vitesse maximale, le robot à une consommation de 280 W. Il lui faut 25 000 seconde pour parcourire 1 km, soit 700 kJ, ce qui correspond à 0,19 kWh. La plupart des batteries de 48 V trouvable sur le marché on une réserve de plus d' un kWh, et les panneaux solaire utilisée pour les rover martien de la NASA sont capable de fournir jusqu'à 1 kWh, l'énergie n'est donc pas un problème.

Figure 41: Puissance électrique (W) en fonction du temps (s)

f) Difficultées et améliorations possibles

Il mentionné dans la partie 'bloc moteur' que le moteur retenu n'était pas suffisant. En effet, pour rouler à environ 0,3 m/s sur le sol martien, il faut que le système {moteur + réducteur} soit capable de fournir un couple de l'ordre de 63,5 N.m sur les roues. En choisissant un moteur de 63,5 W, il est en théorie possible, avec le bon réducteur, de fournir un tel couple à une vitesse satisfaisante. Cependant, les moteurs réels on un couple de décrochage, c'est à dire un couple résistant à partir duquel le moteur cale. Tous les moteurs trouvés sur internet avait un couple de décrochage trop faible.
Pour résoudre ce problème, il faudrait au choix :

- Augmenter le rapport de réduction et utiliser un moteur plus puissant.
- Utiliser un moteur avec un couple de décrochage plus élevé.
- Répartir le poid du véhicule sur plus de roues. Notre moteur pleine puissance permet de rouler à 0,4 m/s avec un couple résistant de 6,35 N.m. Il est donc possible d'utiliser 10 moteurs à pleine puissanc sur 10 roues motrices différentes pour contrer le couple de résistance totale de 63,5 N.m. Dans ce cas, l'énergie consommée serait dix fois plus importante. Même dans ce cas, il existe de nombreuses batteries sur le marché capable de fournir assez d'énergie pour que le robot reste autonome 1h ou plus.

Un deuxième point d'amélioration est la simulation elle même. La résistance au roulement est valide à vitesse constante, pour un terrain parfaitement plat. Si on veut mieux simuler la partie dynamique du mouvement plus précisément, et prendre en compte la présence de pentes sur le sol martien, il faut simuler l'intégralité du véhicule (arbre de transmission, suspension, etc...). Ce genre de simulation est possible avec les toolbox *Simscape Driveline* ou *Simscape Mulitbody*, mais elle dépasse nos compétence en mécanique.

Enfin, pour simplifier la simulation, la position parcourue est obtenue en intégrant la vitesse de rotation des roues. Cette mesure n'est pas fiable si les roues n'adhèrent pas au sol. On pourrait à la place mesurer l'accélération du véhicule à l'aide d'un accéléromètre, puis l'intégrer deux fois pour obtenir la position. Cette mesure permettrait en plus de vérifier que le robot ne dévie pas. Il faut faire attention à ne pas placer l'accéléromètre au niveau du centre de rotation, pour pouvoir mesurer l'accélération centripète. On peut ainsi retrouver la vitesse des roues lorsque le robot tourne sur lui même

V - Capteurs

Etant donné les circonstances du projet, et étant donné que notre équipe n'est constitué que de trois personne, il n'a pas été possible d'étudier pleinement la partie 'Capteurs'. Nous avons cependant trouvé des références de capteurs qui correspondent au cahier des charges.

Pour le capteur d'humidité, on peut en utiliser un similaire au capteur d'humidité relative de REMS présent sur le rover *Curiosity*. Manufacturé par Vaisala, Inc, il possède trois têtes de capteurs HUMICAP®, qui sont des capteurs polymères capacitifs à couches minces déposées entre deux électrodes conductrices.Ce modèle ayant fait ses preuves par le passé, il semble donc cohérent d'opter pour une solution similaire.

Il s'agit d'un capteur mesurant l'humidité relative entre 0 et 100% avec une résolution de 1%. Il peut survivre à des températures allant de -123 à 127°C, mais fonctionne essentiellement dans la gamme -70 à 50°C, qui est incluse dans celle du cahier des charges. Il est de plus doté d'un filtre anti-poussière et a une précision de 8% à -70°C. La capacité du capteur change avec l'humidité, si le micro ordinateur utilisé pour contrôler le robot peut directement mesurer une tension, il peut mesurer directement l'humidité. Il n'y a donc pas forcément besoin de connecter le capteur par liaison Bluetooth ou I2C. Sa consommation énergétique typique vaut 0,15 W, ce qui est négligeable devant la consommation du moteur.

Pour le capteur thermique, la page 'Capteur de température' postée par Marie Drevet-Mulard propose le capteur en platine *P0K1.202.3FW.A.007.* Sa résistance varie linéairement avec la température dans la plage spécifiée dans le cahier des charges. Le post précise que la variation de tension induite par la variation de température est trop faible pour être mesuré avec la carte nucleo, il faut donc amplifier le signal avec un amplificateur différentiel.

Pour la mesure de la distance, on peut utiliser l'accéléromètre ADXL 350, qui communique avec la carte nucleo selon le protocole I2C. Un tutoriel expliquant comment l'utiliser est trouvable sur le site du LEnsE.

VI - Conclusion

Bien que notre étude ne soit pas complète, elle présente bien qu'elle sont les points importants pour la construction du robot et les points à approfondir pour le finaliser. Il est possible qu'il fasse repenser la structure du véhicule pour qu'il puisse se déplacer sur la surface de Mars, et il faut encore réfléchir au montage électrique des capteurs, ainsi qu'au programmes utilisées pour interpréter ses mesures, mais les programmes déjà développés sont conçues pour être utilisables quels que soient les modifications.

Nous n'avons pas eu le temps de prendre en compte la fonction permettant de contrôler la vitesse maximale du robot ('**M_Speed'** et **'MS_slider'**) dans la boucle d'asservissement du moteur. Une manière simple de la mettre en place serait d'insérer un gain variable dans la boucle. Cependant cette méthode ne prend pas en compte les variations de terrain, qui entraînerait des variations de résistance au roulement et donc de vitesse. Une autre méthode plus complexe consiste à asservir la vitesse et la position indépendamment.

Par ailleur le système de communication n'a pas été testé avec les autres composantes de ce projet, cependant, une conception modulaire de chacune des parties du robot nous permet de dire que cela ne posera pas de problèmes. De plus il est encore à implémenter la routine de communication où le robot et l'utilisateur échange des messages régulièrement. Ces détails pourraient être réglée aisément avec un peu de temps supplémentaire.