Première NSI

Travaux Pratiques : Traitement des trames GPS avec python

I. Présentation

1. Objectif

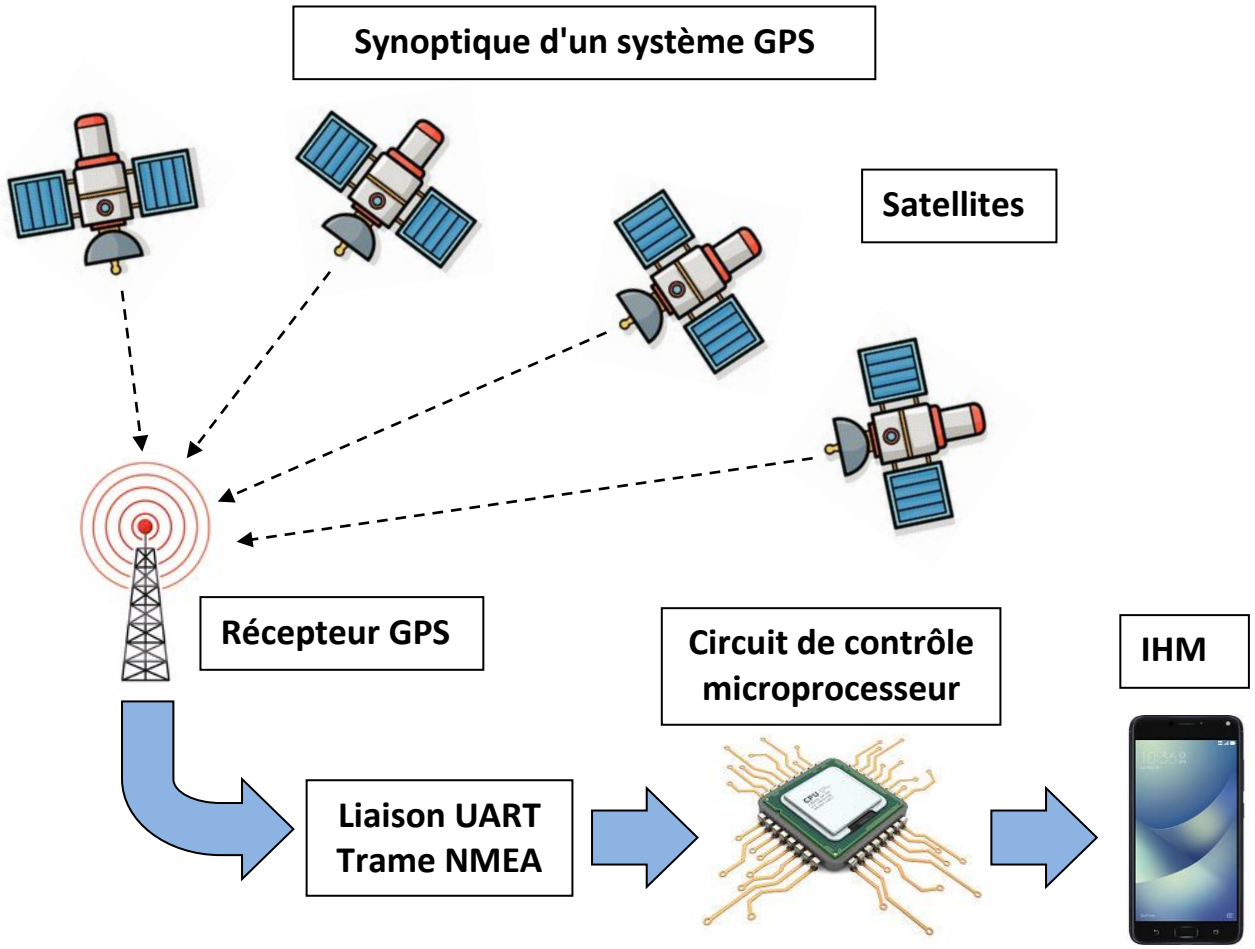
Ce TP a pour but de comprendre le traitement des chaînes de caractères avec python et de mettre en place une communication série avec un port USB du PC.

2. Présentation du système GPS.

Le **Global Positioning System** (**GPS**) (en français : « Système mondial de positionnement ») est un système de positionnement mis en service par le gouvernement américain à des fins militaires. Il a été ouvert aux applications civiles en 2000. Ce système repose sur l'utilisation de 24 satellites qui envoient vers la Terre l'heure de leur horloge atomique embarquée. Un utilisateur, sur Terre qui reçoit ces signaux d'au moins quatre satellites peut, en étudiant le décalage temporel des signaux reçus, connaître sa position avec une précision de quelques mètres.

Ce système est utilisé par les navigateurs type TomTom ou par les téléphones portables qui intègrent pratiquement tous des circuits chargés de recevoir et de décoder les signaux issus des satellites. Ces circuits s'appellent des 'GPS engine'. Ces circuits entièrement intégrés gèrent intégralement la réception ainsi que tous les calculs nécessaires au positionnement. Ils dialoguent avec l'extérieur par une communication série de type UART.

Il existe plusieurs protocoles de communication pour les GPS. Certains sont standardisés (ouverts et utilisables par tous) d'autres sont propriétaires (conçus et utilisés par une seule marque). Le plus courant de ces protocoles s'appelle le **NMEA** qui signifie **National Marine Electronics Association**. Il s'agit d'une association à but non lucratif fondée par un groupement de professionnels, de l'industrie de l'électronique des périphériques marins, conjointement avec des fabricants, des distributeurs, des revendeurs, des institutions d'enseignement. Leur but entre autres, est d'harmoniser et standardiser les équipements de la marine.



3. Le module GP-635T

Le circuit spécialisé GP-635T est un module GPS miniature, avec antenne intégrée. Dès qu'il est alimenté sous 5V, il recherche les satellites pour décoder leurs informations et positionner le module. Il émet alors, via une liaison série, des trames contenant les informations recueillies sur le réseau GPS : heure, date, longitude, latitude, vitesse, etc. Ces informations sont ensuite traitées par le système utilisant le module GPS : téléphone portable, appareils de localisation routière, etc pour en extraire les données utiles. Le module GP-635T ne nécessite aucune configuration pour fonctionner.

4. Le standard NMEA-0183

Sous ce standard, toutes les données sont transmises sous la forme d'une chaîne de caractères ASCII. Chaque trame est normalisée et constituée de la façon suivante :

- ✓ Commence par le caractère '**$**'
- ✓ suivi par deux lettres pour identifier le récepteur visé. Ici, ce sera **GP** pour GPS.
- ✓ puis, un groupe de 3 lettres qui permet d'identifier la trame et son contenu. Il existe plusieurs trames correspondant à des besoins différents. Chaque trame possède une syntaxe différente.
  - **GGA** : pour GPS Fix et Date
  - **GLL** : pour Positionnement Géographique Longitude - Latitude
  - **GSA** : pour DOP et satellites actifs

- **GSV** : pour Satellites visibles
- **VTG** : pour Direction (cap) et vitesse de déplacement (en nœuds et km/h)
- **RMC** : pour données minimales exploitables spécifiques

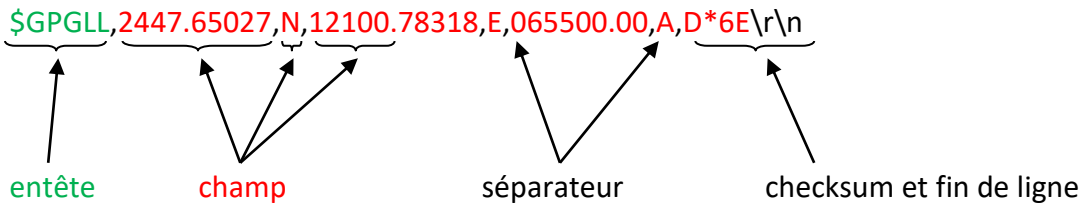
✓ suivi d'un checksum (qui permet de vérifier la cohérence de la trame).

✓ et termine par les caractères retour chariot '\r' et fin de ligne'\n'.

5. Un exemple de trame

Intéressons-nous à une trame souvent utilisée pour connaître la position courante du récepteur : la trame **GLL**. Une trame est constituée de **champs**. Les champs sont séparés entre eux par des **virgules**. Un champ peut être vide mais la présence de la virgule est obligatoire.

Exemple :



Explication :

Contents	Example	Unit	Explanation
Message ID	$GPGLL		GLL protocol header
Latitude	2447.65027		ddmm.mmmmm dd: degree, mm.mmmmm: minute
North/South	N		N: North Latitude, S: South Latitude
Longitude	12100.78318		dddmm.mmmmm dd: degree, mm.mmmmm: minute
East/West	E		E: East Longitude, W: West Longitude
UTC Time	065500.00		hhmmss.ss hh: hour, mm: minute, ss: second
Status	A		A: Data valid, V: Data invalid
Mode Indicator	D		A: Autonomous, D: DGPS, E: DR
checksum	*6E		

figure 1

II. Acquisition des trames : Liaison série

Connecter le module GPS sur un port USB de votre PC.

On fournit la fonction **serialPorts()** qui retourne une liste contenant les ports USB ouverts et disponibles sur votre PC.

Q1 Tester la fonction **serialPorts()** et indiquer ce qu'elle vous retourne

```
['COM5', 'COM15']
```
ou un seul port suivant le PC.

Dans le datasheet du module **GP-635T** on trouve l'indication suivante :

> The default output sentences are GPGGA, GPGSA, GPGSV, GPRMC, GPVTG and GPGLL. The default UART communication parameters are 9600 bps, 8 data bits, 1 stop bit, and no parity. Other baud rate and related configurations could be requested based on MOQ.

Q2 Relevez la fréquence de transmission du module. Indiquez l'unité utilisée.

9600 bauds

L'instruction Serial () permet d'établir la communication avec un port de votre PC. Il faut importer cette fonction depuis la bibliothèque serial. Elle retourne les caractéristiques de la liaison qui seront stockées dans une variable (ici s qui sera utilisée ensuite).

Exemple pour établir la liaison avec le port COM3, è une fréquence de 115200 bauds, avec un time out de 5s.

from serial import Serial

s = Serial('COM3',115200)

Le port ouvert peut être retourné par l'instruction **s.portstr** On peut ensuite lire le port qui vient d'être ouvert en appelant la fonction s.readline. Cette fonction lit tous les caractères entrant jusqu'aux caractères fin de ligne et retour chariot "\r\n" :

reception = s.readline()

Q3 Proposez une fonction nommée **connectionGPS()** qui permet de se connecter sur le port libre. Cette fonction devra retourner la variable s contenant les caractéristiques de la liaison série.

Amélioration possible de fonction : demander à l'utilisateur le port qu'il souhaite dans le cas où la liste de ports disponibles comporte plusieurs noms. Ajout d'un timeout. Prévoir une sortie dans le cas où la connexion échoue.

```
##############################################################################
##                    Configuration de la liaison UART                      ##
##############################################################################
def connectionGPS_V0():

    ports = serialPorts()

    print(ports)

    port = ports[0]

    s = Serial(port,9600)

    print("UART opérationnel sur", s.portstr, '\n')

    return(s)
```

```
##########################################################################
##                    Configuration de la liaison UART                  ##
##########################################################################
def connectionGPS_V1():
    ports = serialPorts()
    #si un seul port dispo, c'est parti
    if len(ports) == 1:
            port = ports[0]
    else:
            #sinon, il faut demander
            port = input("Saisissez un autre port " + str(ports) + " : ")

    while 1:
            print ('Essai de connection sur le port', port, "... ",end="")
            try:
                    #creation de l'objet UART avec un timeout de 2s (indispensable, sinon risque de blocage)
                    s = Serial(port,9600, timeout=2)
                    print("UART opérationnel sur", s.portstr, '\n')
                    return(s)
            except:
                    print("échec !")
                    ports = serialPorts()
                    port = input("Saisissez un autre port", str(ports), ":")
```

Q4 Ecrivez un programme qui se connecte au port et lit en boucle tout ce qui arrive sur le port et l'affiche.

Programme :

```
##########################################################################
##                          Corp du programme                           ##
##########################################################################

ser = connectionGPS_V0()

while 1:

    reception = ser.readline()
    print(reception)
```

Affichage :

```
b'$GPGSA,A,3,10,27,18,16,08,20,,,,,,,2.38,1.47,1.87*07\r\n'
b'$GPGSV,4,1,15,01,27,259,12,07,01,266,,08,78,296,18,10,49,064,08*70\r\n'
b'$GPGSV,4,2,15,11,36,279,,14,05,132,,16,14,177,21,18,51,266,20*7F\r\n'
b'$GPGSV,4,3,15,20,25,050,12,21,01,076,,22,14,207,,27,60,125,19*77\r\n'
b'$GPGSV,4,4,15,28,09,327,,30,05,294,,32,15,117,16*44\r\n'
b'$GPGLL,4827.70626,N,00006.78664,E,150631.00,A,A*68\r\n'
b'$GPRMC,150632.00,A,4827.70524,N,00006.78631,E,2.093,,280619,,,A*7F\r\n'
b'$GPVTG,,T,,M,2.093,N,3.876,K,A*21\r\n'
b'$GPGGA,150632.00,4827.70524,N,00006.78631,E,1,06,1.47,157.4,M,45.9,M,,*54\r\n'
b'$GPGSA,A,3,10,27,18,16,08,20,,,,,,,2.38,1.47,1.87*07\r\n'
b'$GPGSV,4,1,15,01,27,259,12,07,01,266,,08,78,296,18,10,49,064,14*7D\r\n'
b'$GPGSV,4,2,15,11,36,279,,14,05,132,,16,14,177,22,18,51,266,20*7C\r\n'
b'$GPGSV,4,3,15,20,25,050,11,21,01,076,,22,14,207,,27,60,125,19*74\r\n'
b'$GPGSV,4,4,15,28,09,327,,30,05,294,,32,15,117,16*44\r\n'
b'$GPGLL,4827.70524,N,00006.78631,E,150632.00,A,A*6A\r\n'
b'$GPRMC,150633.00,A,4827.70447,N,00006.78501,E,2.465,,280619,,,A*77\r\n'
b'$GPVTG,,T,,M,2.465,N,4.565,K,A*24\r\n'
b'$GPGGA,150633.00,4827.70447,N,00006.78501,E,1,06,1.47,157.5,M,45.9,M,,*50\r\n'
```

Q5 Indiquez le type de la variable **reception**.

type 'bytes' : il s'agit en fait d'une liste d'octets codés en ASCII. C'est un objet mais on n'est pas obligés de le dire.

Les trames reçues sont encodées en ASCII. Pour traiter plus facilement ces données, il est nécessaire de les encoder en unicode (UTF-8) qui est le système codage des chaînes de caractères utilisées par python. Il faut utiliser l'instruction **decode** :

textRecu = reception.decode('utf8')

Q6 Ajouter cette instruction à votre programme, afficher le résultat, comparer avec ce qui s'affichait précédemment et indiquez le type de la variable textRecu.

```
$GPGSV,4,4,15,28,09,327,,30,05,294,,32,15,117,15*47
$GPGLL,4827.70691,N,00006.78083,E,150604.00,A,A*6D
$GPRMC,150605.00,A,4827.70758,N,00006.78340,E,4.182,59.63,280619,,,A*51
$GPVTG,59.63,T,,M,4.182,N,7.745,K,A*0A
$GPGGA,150605.00,4827.70758,N,00006.78340,E,1,06,1.47,156.5,M,45.9,M,,*5A
$GPGSA,A,3,10,27,18,16,08,20,,,,,,,2.39,1.47,1.88*09
$GPGSV,4,1,15,01,26,259,12,07,01,266,,08,78,296,16,10,49,064,12*74
$GPGSV,4,2,15,11,35,279,,14,05,133,,16,15,177,20,18,51,266,18*76
$GPGSV,4,3,15,20,25,050,15,21,01,076,,22,14,207,,27,60,125,14*7D
```

le préfixe b'' qui indiquait la liste d'octets a disparu. Il s'agit maintenant d'un type 'str'

Q7 En utilisant le document en annexe qui indique le contenu de chaque trame NMEA, trouver celle qui contient :

- ✓ L'heure UTC
- ✓ la date
- ✓ la longitude
- ✓ la latitude

Repérez les différents champs contenant ces données sur cette trame.

la seule qui contient tout ça est la trame **RMC**.

```
$GPRMC,150606.00,A,4827.70821,N,00006.78666,E,5.079,62.16,280619,,,A*5C
```

Pour faciliter le traitement des trames, on peut les convertir en une liste qui contiendra tous les champs de notre trame. Il existe une instruction qui permet de couper un texte en utilisant un élément séparateur.

textRecu.split('élément séparateur')

Cette instruction retourne une liste.

Q8 Dans notre cas, indiquer l'élément séparateur des différents champs de notre trame.

Il s'agit d'une virgule

Q9 Proposez une fonction nommée **NmeaToList()** qui lit le port série et retourne une liste contenant les différents champs de notre trame.

Amélioration possible de fonction : placer l'instruction de décodage dans une structure try except. Cela évite de planter le programme ; la trame reçue est vide par exemple.

```
##############################################################################
##                    Récupération des trames GPS en liste                  ##
##############################################################################
def NmeaToList(s):

    bytesRecu = s.readline()      #lit jusqu'au fin de ligne

    try:
        #decodage en unicode
        #placé dans un try car cette instruction plante si la trame recue est vide
        textRecu = bytesRecu.decode('utf8')
    except:
        textRecu = ""

    return(textRecu.split(','))
```

```
['$GPRMC', '155616.00', 'A', '4827.71236', 'N', '00006.78740', 'E', '0.531', '', '280619', '', '', 'A*71\r\n']
['$GPVTG', '', 'T', '', 'M', '0.531', 'N', '0.983', 'K', 'A*26\r\n']
['$GPGGA', '155616.00', '4827.71236', 'N', '00006.78740', 'E', '1', '05', '2.75', '145.0', 'M', '45.9', 'M', '', '*53\r\n']
['$GPGSA', 'A', '3', '22', '27', '18', '08', '01', '', '', '', '', '', '', '', '5.09', '2.75', '4.28*05\r\n']
['$GPGSV', '3', '1', '12', '01', '46', '276', '15', '03', '14', '214', '', '08', '75', '167', '25', '10', '31', '051', '*77\r\n']
['$GPGSV', '3', '2', '12', '11', '57', '289', '13', '14', '22', '118', '13', '18', '73', '295', '21', '20', '05', '049', '*7D\r\n']
['$GPGSV', '3', '3', '12', '22', '36', '212', '22', '27', '38', '139', '21', '28', '23', '315', '23', '32', '30', '099', '*77\r\n']
['$GPGLL', '4827.71236', 'N', '00006.78740', 'E', '155616.00', 'A', 'A*6B\r\n']
```

Q10 Ecrivez un programme qui teste s'il s'agit de la trame souhaitée et qui affiche les champs décrits à la question Q7.

```
ser = connectionGPS_V0()

while 1:

    listGPS = NmeaToList(ser)

    #détection d'une trame type $GPRMC qui contient les infos souhaitées
    if listGPS[0] == "$GPRMC":
        print(listGPS[1])
        print(listGPS[3])
        print(listGPS[4])
        print(listGPS[5])
        print(listGPS[6])
        print(listGPS[9])
        print("")
```

Affichage :

```
162700.00
4827.71571
N
00006.77999
E
280619
```

Q11 On souhaite maintenant mettre en forme les données pour rendre leur lecture plus agréable. Ecrivez deux fonctions nommées extraireHeure(str) et extraireDate(str) qui reçoivent en paramètre respectivement le champ contenant l'heure et celui contenant la date. Ces fonctions retournent des chaînes de caractères contenant l'heure et la date, comme ci dessous :

16h28m57s

28/06/2019

De sorte que l'instruction suivante :

print("Heure UTC",extraireHeure(listGPS[1]),"le",extraireDate(listGPS[9]))

doit afficher :

Heure UTC 16h28m57s le 28/06/2019

Remarque : Les syntaxes **texte[:i]**, **texte[i:j]**, et **texte[:j]** permettent de couper une texte en sélectionnant le début, le milieux, ou la fin du texte, entre les caractères i et j.

Amélioration possible : utiliser try except pour éviter que le programme ne plante si une trame est incomplète (champ vide).

```
################################################################################
##                          Extracion des données                             ##
################################################################################
def extraireHeure(strHeure):
    try:
        #strHeure est de la forme "HHMMSS.00"
        #2 premiers caractere vers heure
        heure =  strHeure[:2]
        #2 suivants vers minute
        minute =  strHeure[2:4]
        #2 suivants vers minute
        seconde =  strHeure[4:6]

        #retourne un str mis en forme HH:MM:SS
        return(heure + "h" + minute + "m" + seconde + "s")

    except:
        return("non dispo")

def extraireDate(strDate):
    try:
        #strDate est de la forme "JJMMAA"
        #2 premiers caractere vers jour
        jour =  strDate[:2]
        #2 suivants vers mois
        mois =  strDate[2:4]
        #2 suivants vers minute
        annee =  "20"+strDate[4:6]

        #retourne un str mis en forme HH:MM:SS
        return(jour + "/" + mois + "/" + annee)

    except:
        return("non dispo")
```

Q12 Expliquer comment est codée la latitude dans la trame. Vous trouverez cette info en lisant attentivement la figure 1. Expliquez la signification des minutes d'angle.

Q13 Calculez en décimale l'angle correspondant au champ latitude suivant :

2447.65027

La latitude est donnée sous forme ddmm.mmmmm, donc il s'agit de 24° auquel on doit ajouter 47.65027 minute d'angle (ou 1/60eme de degrés). L'angle vaut donc

Latitude = 24 + 47.65027/60 = 24,79417°

Longitude et latitude sont données sous la même forme.

Q14 Proposez une fonction nommée **extraireLat(strLatLong)** qui reçoit en paramètre d'entrée le champ contenant la latitude et la longitude et renvoie un angle en degrés au format **flottant**. Le programme final doit afficher les données sous la forme suivante :

```
Heure UTC 16h50m12s le 28/06/2019
Latitude :  48.4619 °  N
Longitude :  0.113 °  E
```

```
def extraireLat(strLatLong):
    try:

        return(float(strLatLong[:2]) + float(strLatLong[2:])/60)

    except:
        return(0)
```

```
ser = connectionGPS_V0()

while 1:

    listGPS = NmeaToList(ser)

    #détection d'une trame type $GPRMC qui contient les infos souhaitées
    if listGPS[0] == "$GPRMC":

        #print(listGPS)

        #affichage de l'heure contenu sous forme HHMMSS.00 dans la case 1 de la liste
        print("Heure UTC",extraireHeure(listGPS[1]),"le",extraireDate(listGPS[9]))


        #affichage de la latitude
        print("Latitude : ",round(extraireLat(listGPS[3]),4),"° ",listGPS[4])

        #affichage de la longitude
        print("Longitude : ",round(extraireLong(listGPS[5]),4),"° ",listGPS[6])

        print("")
```

Référence image :

Page 1 : http://www.my-trail.fr/gps-glonass-cest-cest-quoi/

Page 2 :