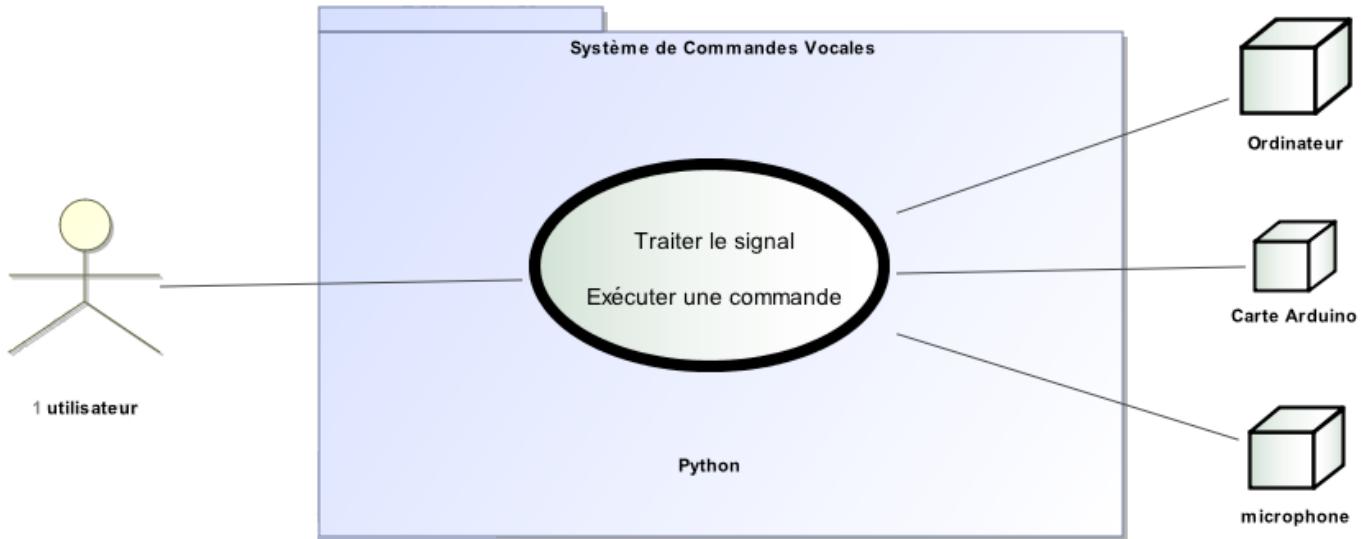


# Traitement du signal : utilisation dans les commandes vocales.

Diagramme de Cas d'utilisation SysML



Plan de la présentation:

## **I – La conversion analogique – numérique**

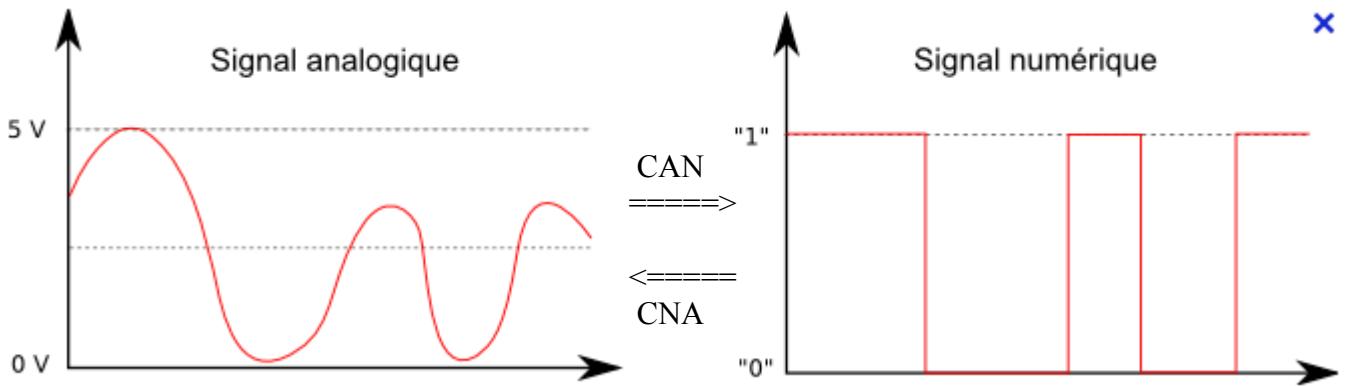
- 1 – principes fondamentaux
- 2 – réalisation d'un modèle de convertisseur : codage d'une tension continue en un signal discret

## **II – Réalisation de la structure de commande vocale**

- 1 – diagramme des exigences
- 2 – traitement algorithmique du signal
- 3 – montage de retranscription du phonème

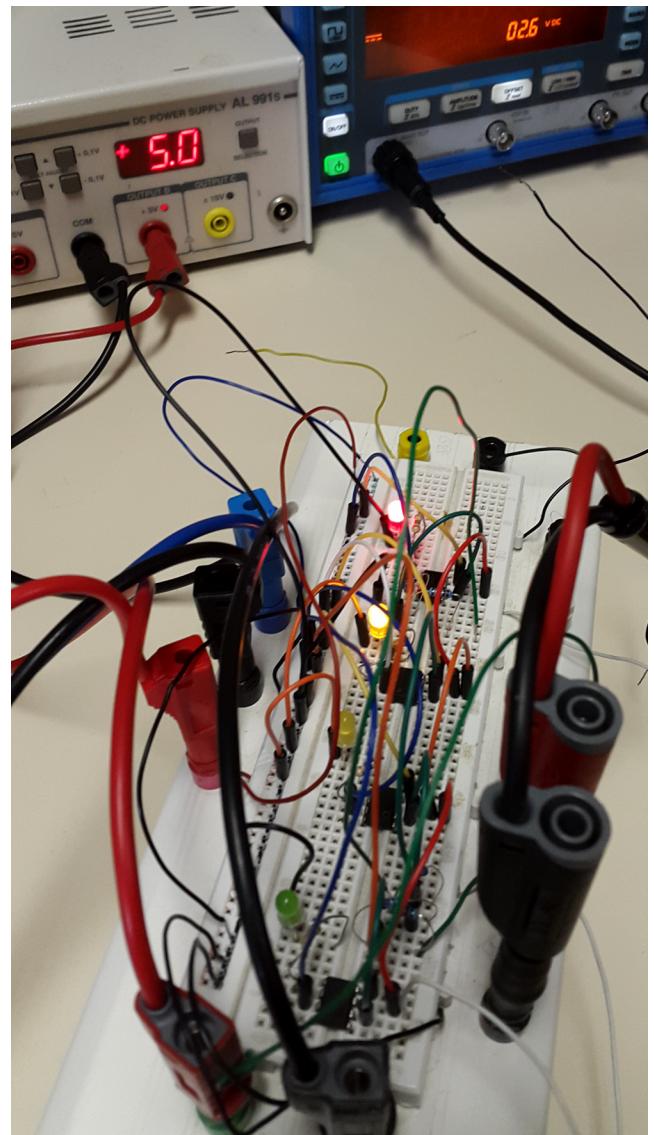
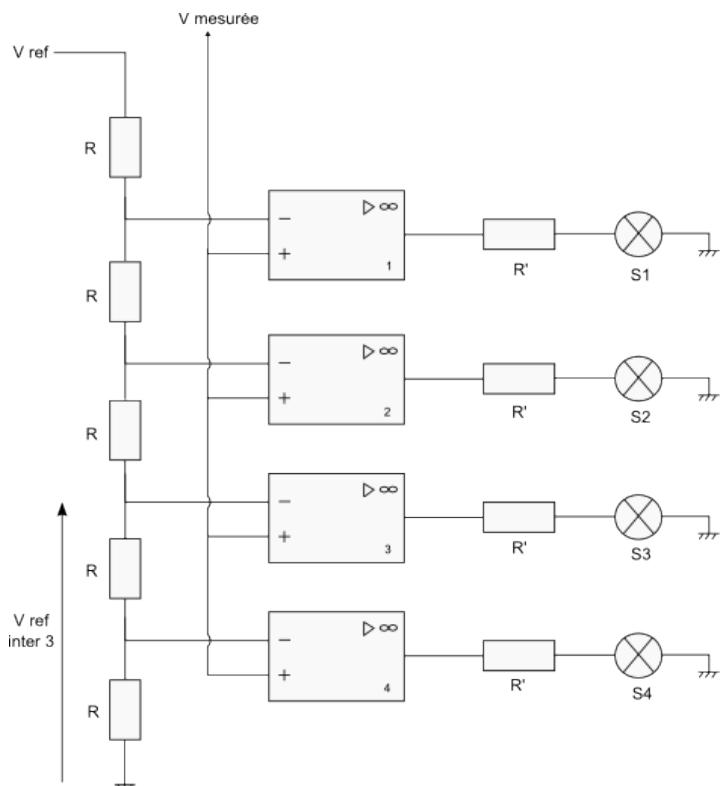
## **III – Conclusions et perspectives envisageables**

## I – La conversion analogique – numérique



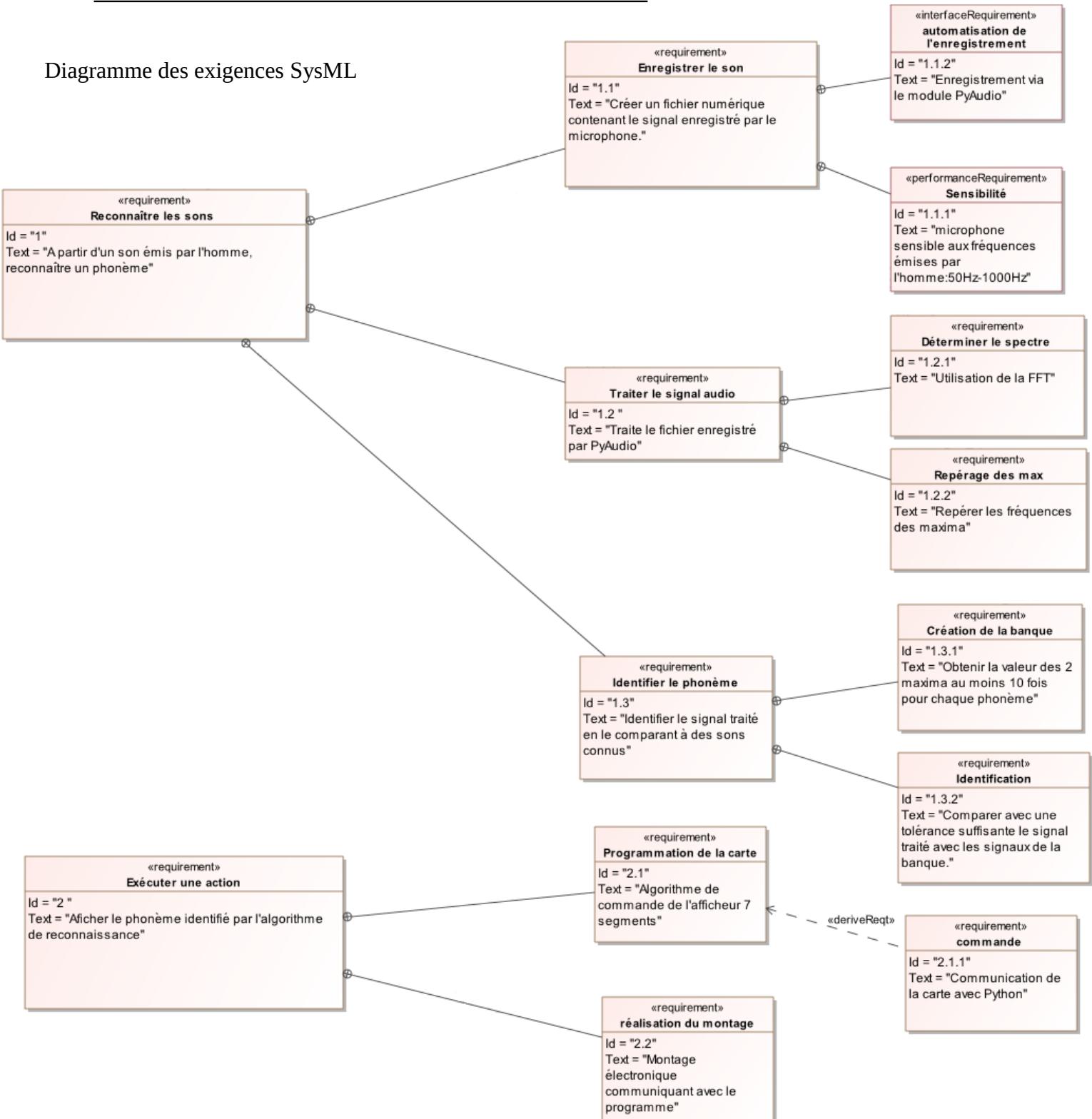
Les CAN sont présents dans un grand nombre d'objets et permettent de numériser un signal analogique.

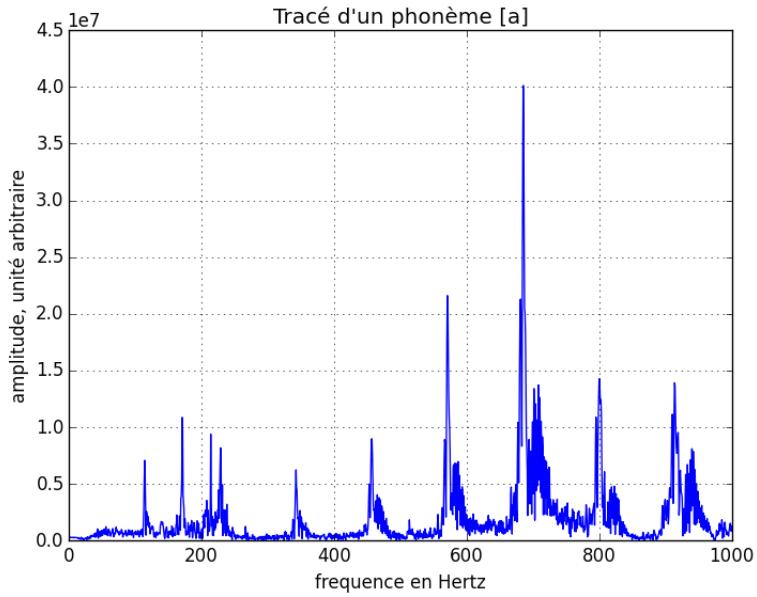
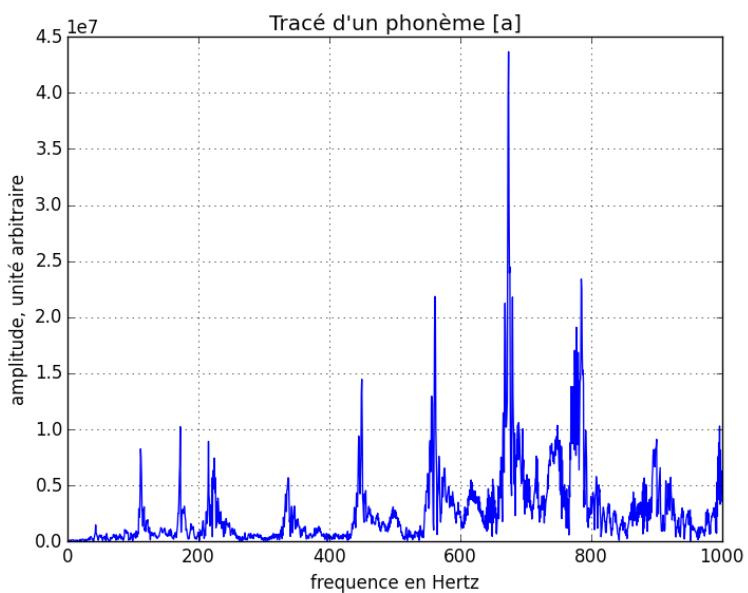
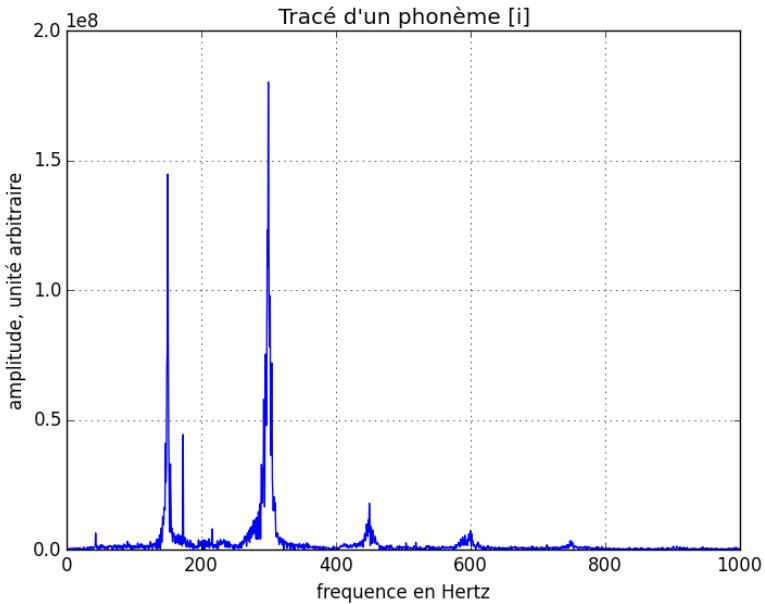
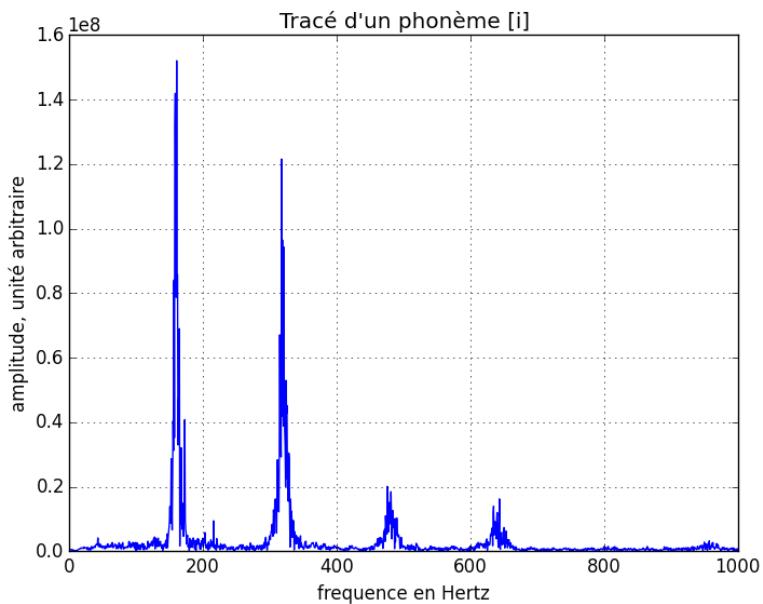
Expérience réalisée : Codage d'une tension continue en électricité :



## II – Réalisation de la structure de commande vocale

Diagramme des exigences SysML





```

416 def reconnaissance2():
417     #permet la reconnaissance d'un phonème
418     nom = input('Entrez votre pseudo: ')
419     if nom == 'etienne':
420         banque = banque_etienne
421     elif nom == 'jb':
422         banque = banque_jb
423     else:
424         print('cest votre première utilisation, il va falloir vous enregistrer')
425         banque = creationBanque()
426     print('données en traitement...')
427     donnee = traitement_banque_moyenne(banque)
428     print('La reconnaissance vocale va pouvoir démarrer')
429     while True:
430         arret = input('si vous voulez arrêter, tapez stop, sinon tapez entrée : ')
431         if arret == 'stop':
432             return ('merci d'avoir utilisé notre service')
433         son = enregistrement('echantillon', 1.5)
434         f1 = fréquence_max1(son)
435         f2 = fréquence_max2(son)
436         listePhoneme = banque[0]
437         n = len(listePhoneme)
438         p = 20 #pourcentage de précision
439         identification = False
440         for i in range(n):
441             d = donnee[i]
442             if (d[0]-t(f1,p)< f1 < d[0]+t(f1,p) and d[1]-t(f2,p) < f2 < d[1]+t(f2,p)) or (d[0]-t(f2,p)< f2 <
443             d[0]+t(f2,p) and d[1]-t(f1,p) < f1 < d[1]+t(f1,p)):
444                 print('vous avez prononcé le son : ', listePhoneme[i])
445                 identification = True
446             if identification == False:
447                 print('signal non identifié')
448                 print(f1,f2)

```

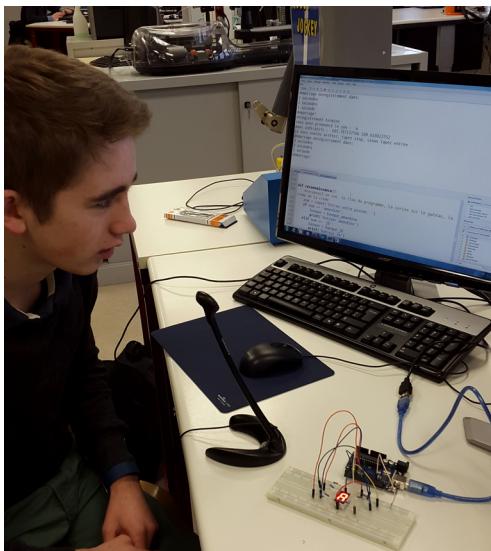
Banque de phonème,  
enregistrée au  
préalable

Comparaison des fréquences identifiées avec celles des  
phonèmes connus

```

35 def instructionArduino(chaine):
36     #lettre à entrer par exemple
37     serial_port.write(str(chaine).encode('ascii')) #envoi de l'instruction, codée en ascii
38     serial_port.readline()#lit ce qu'envoie l'arduino

```



### Structure du montage Arduino :

- l'utilisateur prononce un phonème
- reconnaissance du phonème par le programme
- communication du message à la carte Arduino via Python
- affichage du phonème prononcé sur l'afficheur à diodes

### III - Conclusions et perspectives envisageables

#### Influence du seuil de tolérance sur la réussite du programme :

Expérience réalisée : prononciation du son [i] **25** fois par le même utilisateur

	Tolérance : 0%	5%	20%	60%
issue : [i]	0	16	24	25
[o]	0	0	0	<b>19</b>
[a]	0	0	0	0
« signal non identifié »	25	9	1	0

#### Bilan pour une tolérance de 20% :

Phonème	Taux de réussite
[i]	96%
[o]	92%
[a]	88%

#### Influence de la connaissance préalable de l'utilisateur :

Expérience réalisée : un utilisateur utilise sa banque de données puis celle d'un autre

Comparaison des taux de succès de reconnaissance

	Prononciation : [i]	[o]	[a]
Banque personnelle	96%	92%	88%
Banque d'un autre utilisateur	0%	0%	4%