



A black and white photograph showing a drone planter in flight against a cloudy sky. The drone has four propellers and a central frame with a small wheel. It is positioned above a dense field of young, leafy trees. The foreground is filled with the dark foliage of the trees.

# DRONE PLANTEUR D'ARBRES

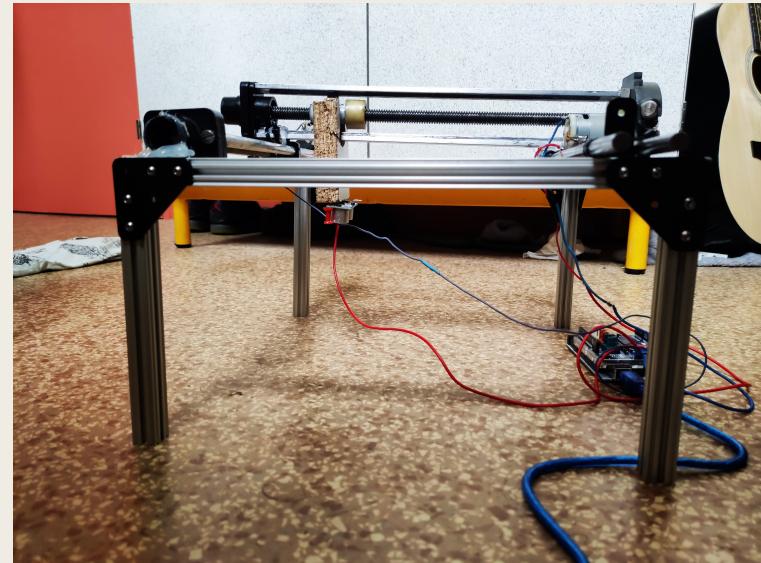
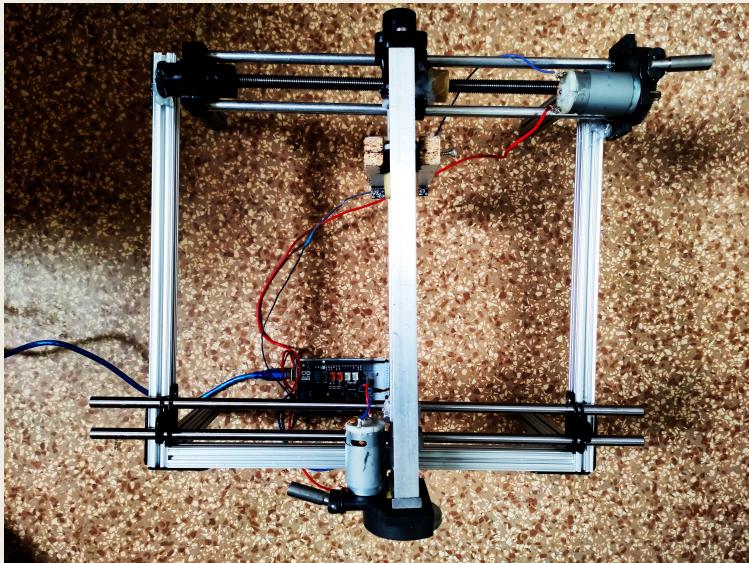
TIPE : Hamza Errahj



## Problématique :

- Comment les drones planteurs d'arbres évoluent-ils dans leurs environnements?

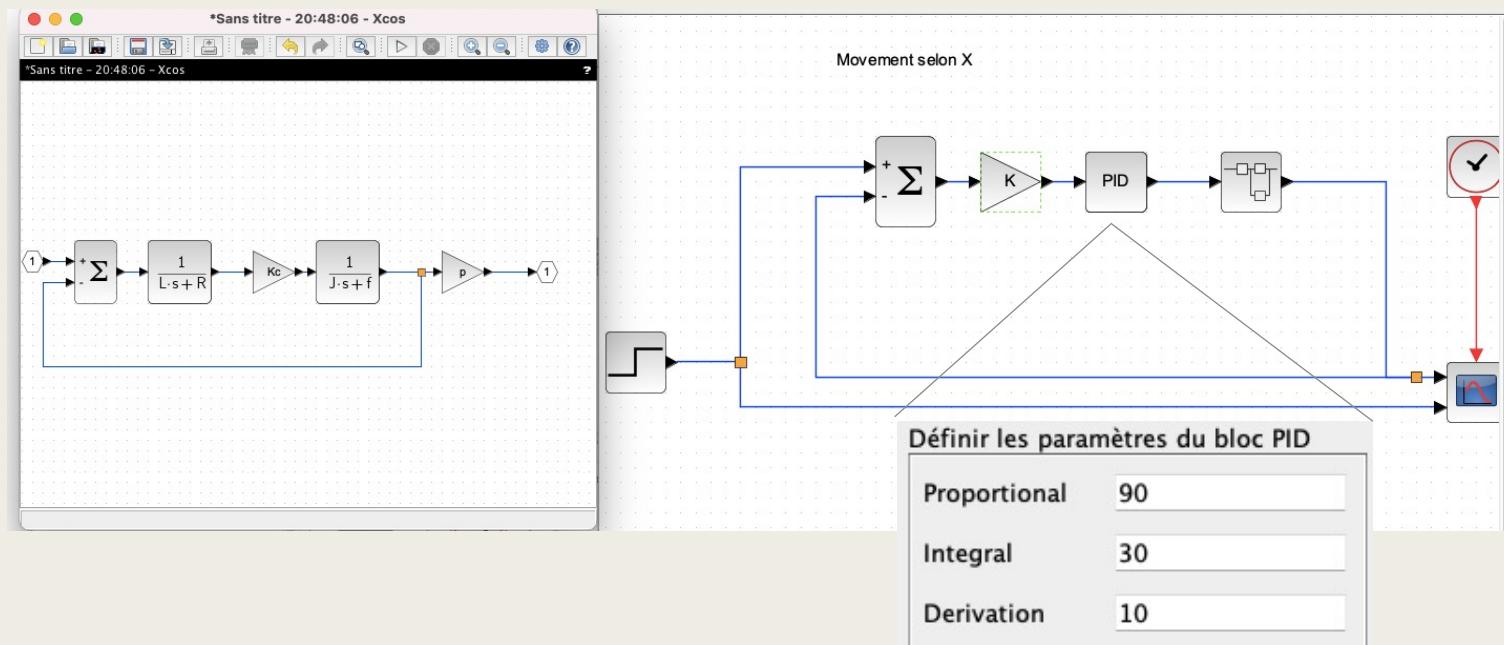
# Modèle 2-axes :



# Cahier des charges :

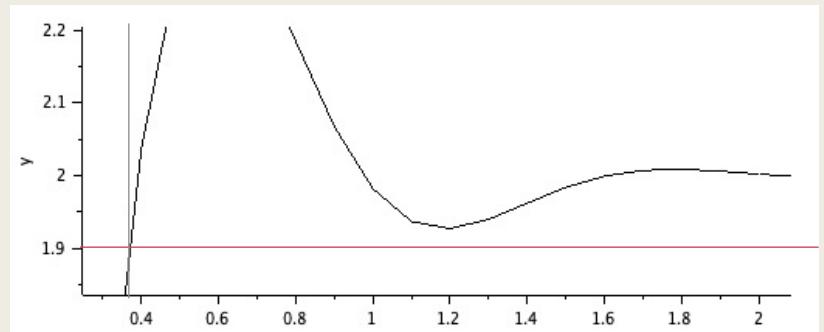
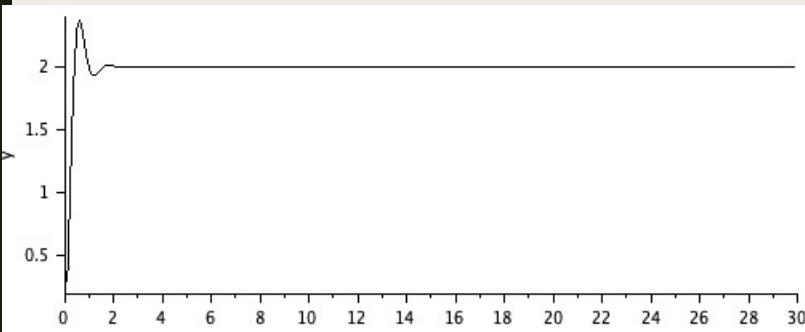
| Fonction                          | Critère   | Niveau   |
|-----------------------------------|---|--|
| Répondre à un échelon de position | <ul style="list-style-type: none"><li>• Temps de réponse</li><li>• Erreur statique</li></ul>                          | <ul style="list-style-type: none"><li>• &lt;0.5s</li><li>• Nulle</li></ul>   |
| Avoir un système autonome         | <ul style="list-style-type: none"><li>• Le système est autonome</li><li>• Scan Rapide</li><li>• Scan précis</li></ul> | <ul style="list-style-type: none"><li>• Ø</li><li>• <math>2 \text{ cm}^2 \cdot \text{s}^{-1}</math></li><li>• Ecart 5%</li></ul> |

# Schéma bloc théorique:



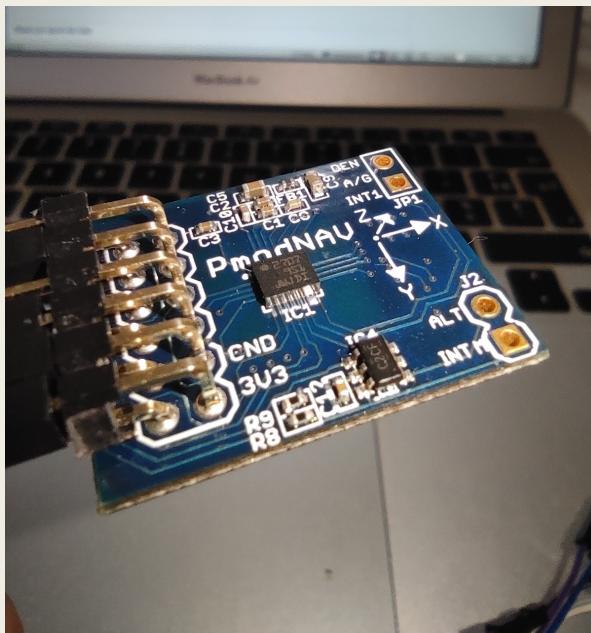
$$J=0.000052 \text{ kg.cm}^2 ; f=0.01 ; K_c=0.235 \text{ N.m.A}^{-1} ; R=2 \Omega ; L=0.23 \text{ H} ; K=1$$

CDC:  $t_{5\%} = 0.38s$   
 $\varepsilon = 0$



Mesure avec un échelon 2

# Centrale inertielle : PmodNAV

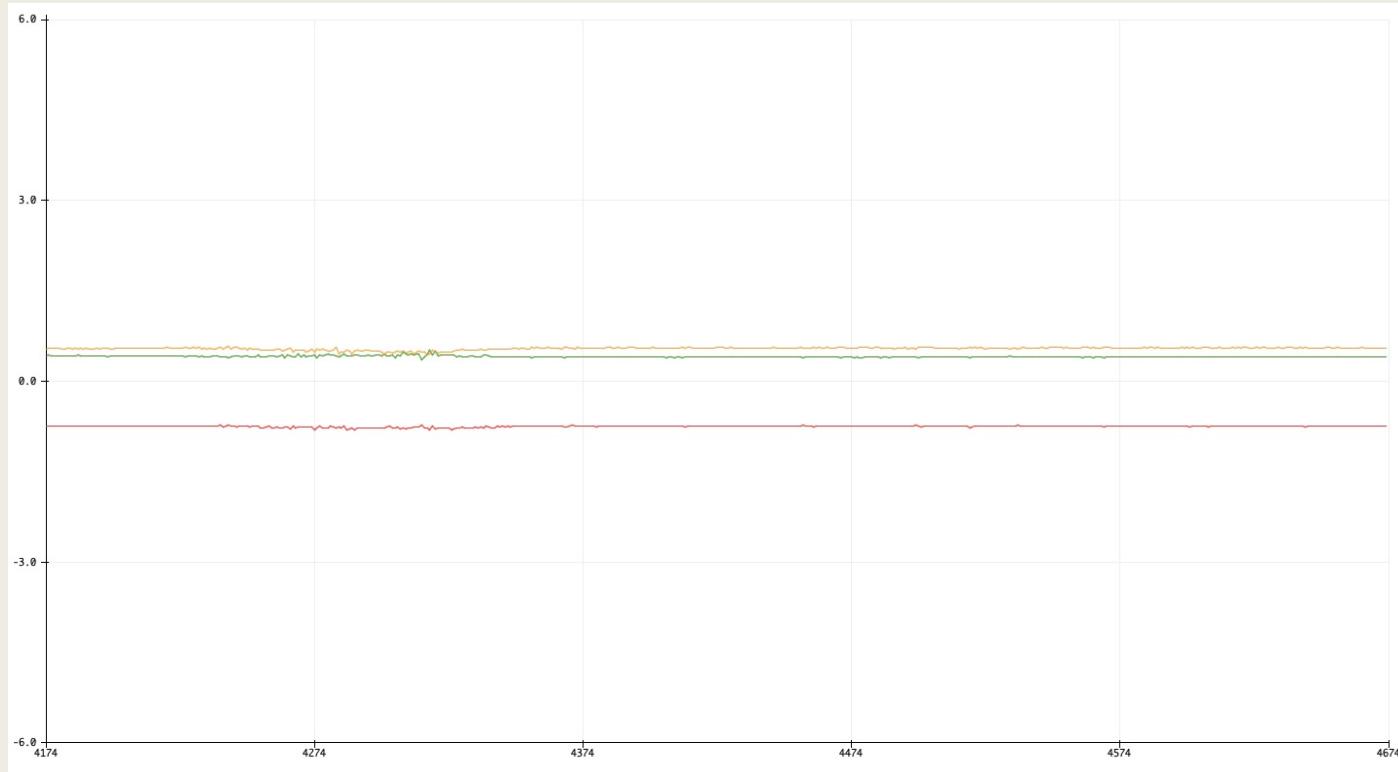


- Constitué d'un accéléromètre, d'un gyroscope et d'un magnétomètre → capteur 9-axes
- Mesure les différentes accélérations (angulaire et linéique)
- On va se concentrer sur la partie accéléromètre

# Bruit :

Défauts :

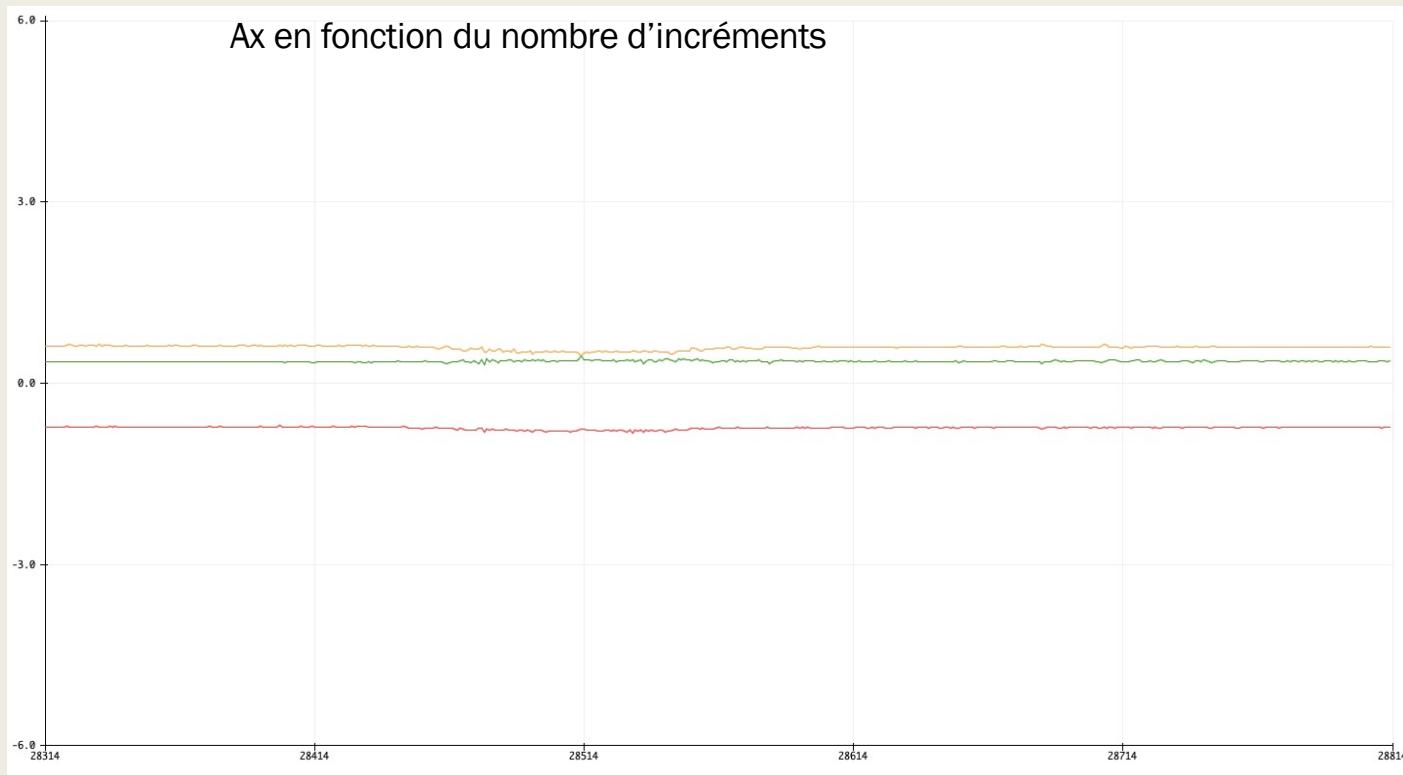
Centrale parfaitement immobile  
Erreur statique non nulle  
Mesure accélération de la pesanteur



Bruit dû à la conception et à l'amplification du signal

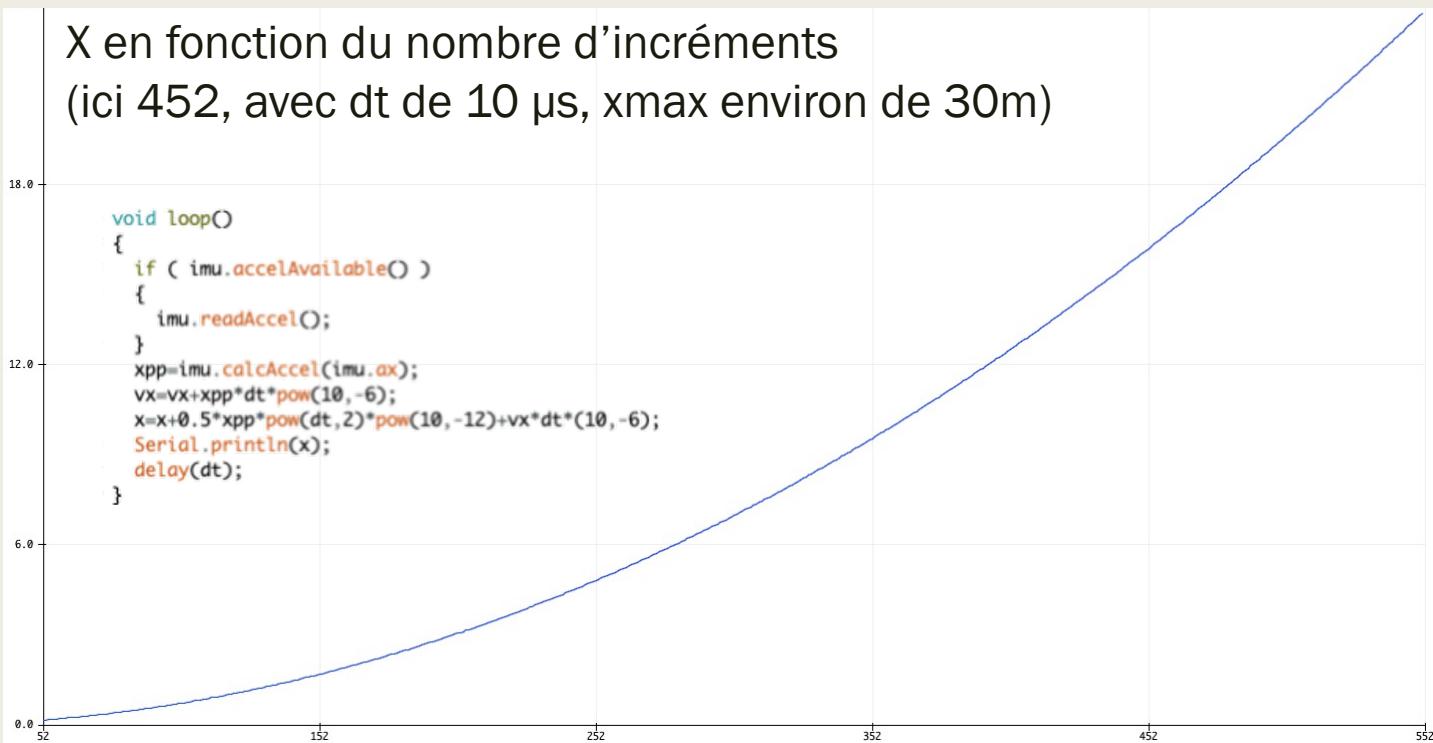
# Bruit :

**Erreur non constante :**  
On ne peut pas la corriger avec un offset



Après 5mn  
sur place,  
sans  
mouvement,  
ni vibration...

# Centrale immobile



# Erreur quadratique de la centrale inertIELLE :

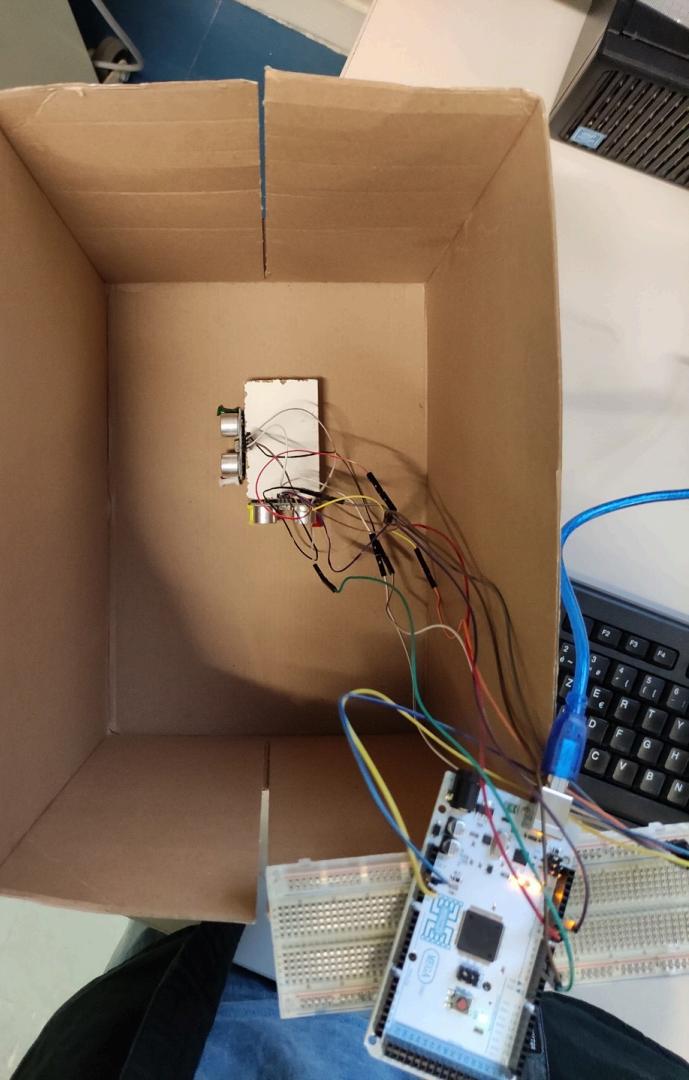
On note  $\epsilon$  l'erreur dûe au bruit - qu'on suppose constante - de l'accéléromètre, avec  $\epsilon$  en  $m.s^{-2}$ . De même on note  $\ddot{x}$  et  $\dot{x}$  respectivement l'accélération et la vitesse selon  $x$ . Le capteur *PmodNAV* renvoie alors  $v = \epsilon + \ddot{x}$  :

$$\text{alors } \int_0^t v(u) \, du = \dot{x} + \epsilon t$$

$$\text{et } \boxed{\int_0^t \int_0^t v(l) \, dl \, du = x + \frac{\epsilon t^2}{2}}$$

d'où l'erreur quadratique du correcteur qui fait diverger vers  $+\infty$ .

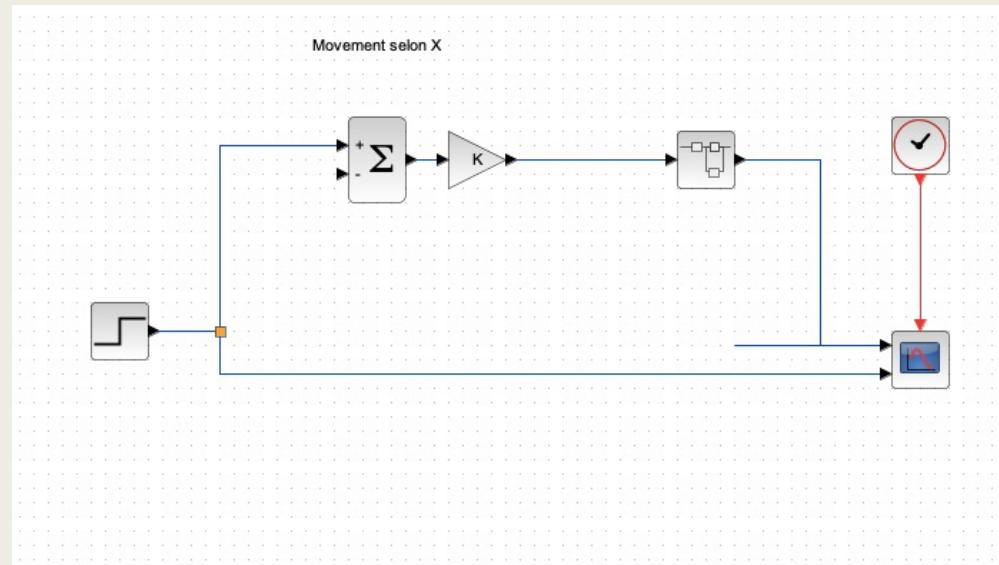
# Détection de la position pour l'asservissement :



- Utilisation d'émetteur/récepteur ultrason pour mesurer la distance entre le “drone” et le mur
- Réimplantation pour le système de sondage
- Impossible à utiliser dans le modèle 2 axes

# Schéma bloc réel :

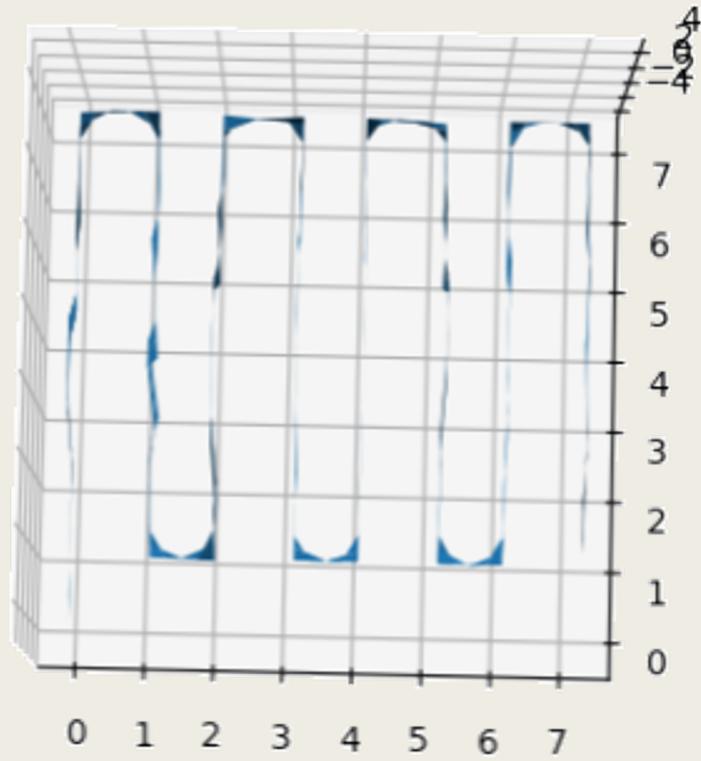
Le capteur de position n'a pas pu être installé : le système n'est pas bouclé



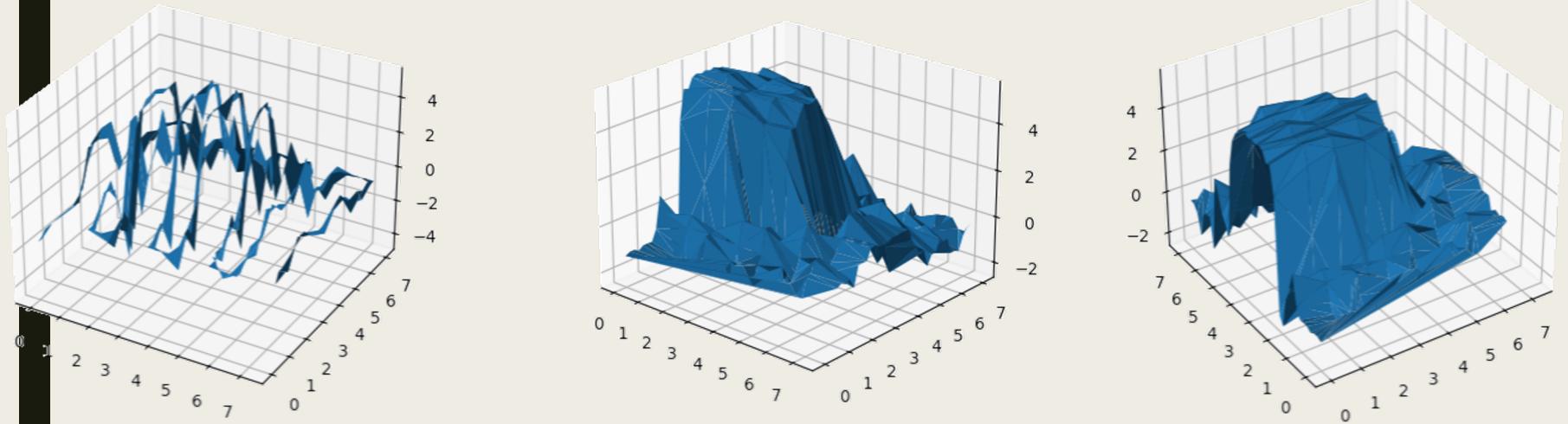
// Code pour PID écrit en annexe



# Expérience



Chemin  
emprunté  
par le  
drone



## Sondage (sans asservissement)

CDC : Temps pour sonder une zone  $49\text{ cm}^2$  de 24 secondes

# Evaluation des écarts entre l'objet sondé et l'objet réel :

- Par python on a un volume de :  $19.21 \text{ cm}^3$
- En réalité on a un volume de :  $18 / 0.3 \text{ cm}^3$

## Interprétation :

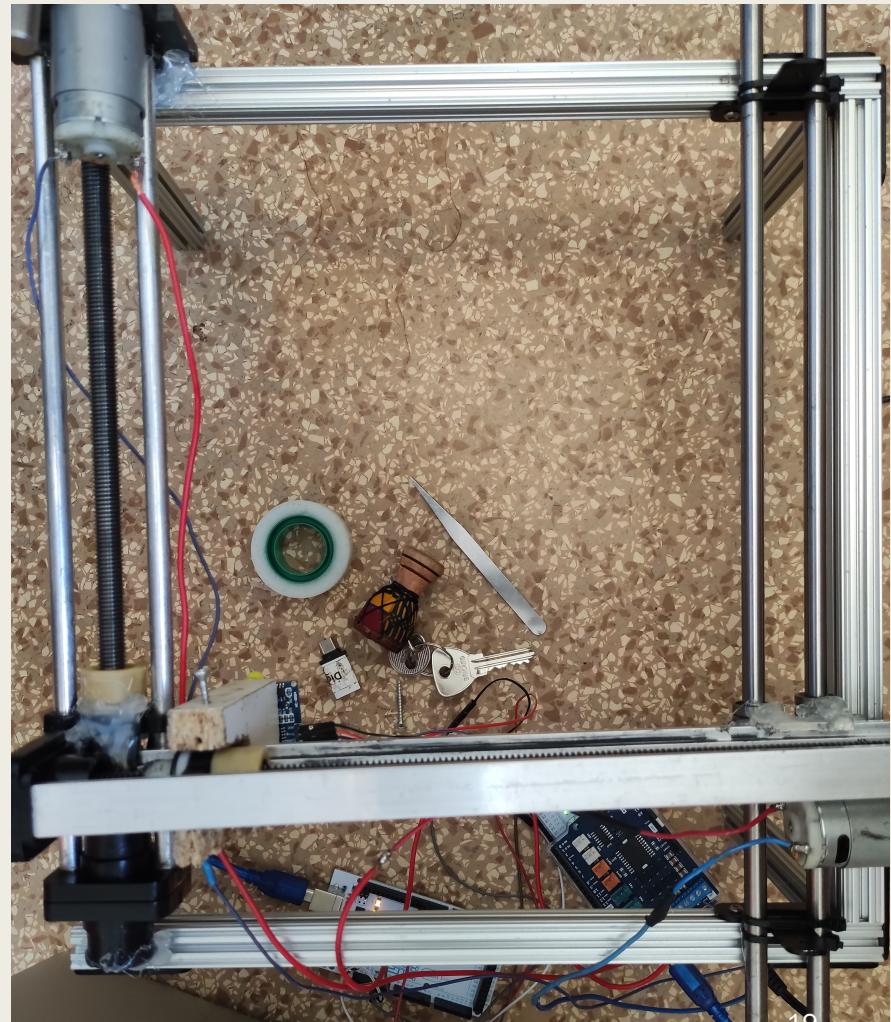
- Ainsi on a un écart de 6,7% pour un système de l'ordre du centimètre
- Dans le cas du système réel, on est dans l'ordre du mètre : l'écart entre le réel et scanné sera inférieur, on valide donc le modèle
- **CDC** : écart inférieur dans le système réel

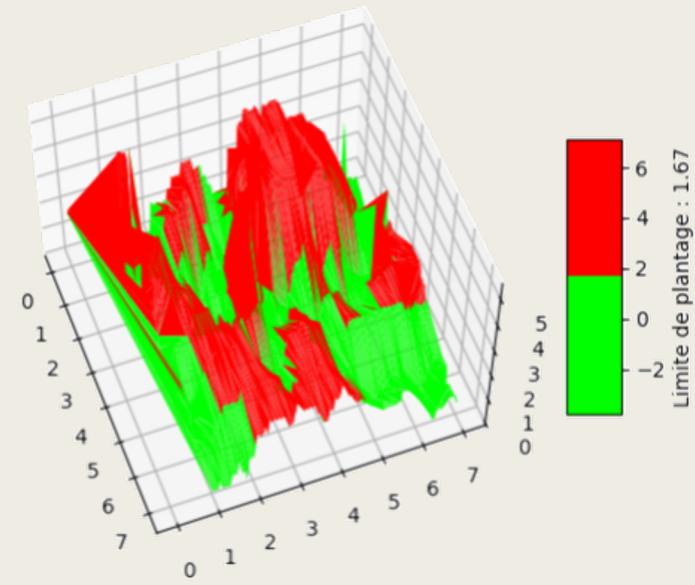
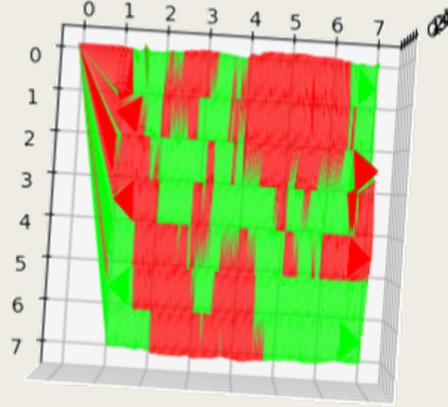
# Interprétation des résultats donnés par le scan 3d :

2 méthodes :

- À chaque point de la figure 3d, on regarde la dérivée selon théta à r fixé (très petit) : si la dérivée dépasse une certaine valeur, le point est un contour de zone non plantable, sinon c'est un point neutre. Si le point est à l'intérieur d'un contour non plantable, il est non plantable.  
À la fin de l'algorithme, tous les points neutres deviennent plantables.
- On fait une intégrale triple, puis on calcule la moyenne : si le point est à une hauteur supérieure à la moyenne, ce n'est pas plantable, sinon il l'est.

# Obstacles pris :





Méthode retenue :  
méthode des moyennes

# Sortie pour un scan de résolution 8x8

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |

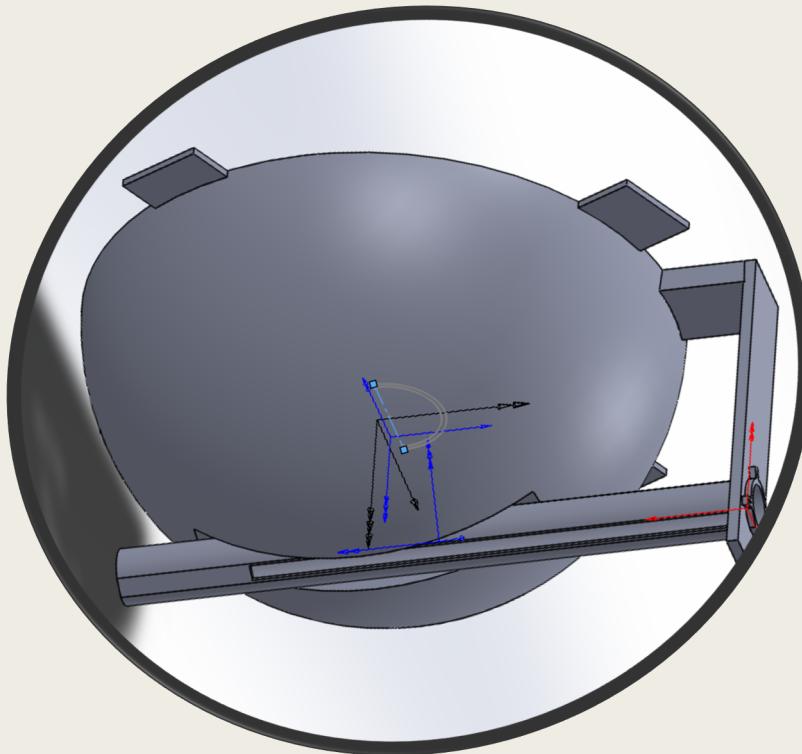
# Synthèse :

Le système est réalisable dans la mesure où on arrive à accéder à la position la plus exacte possible du drone (ici de l'ordre du centimètre)

Plusieurs solutions se présentent pour cela :

- Centrale inertielle (Dérive)
- Module GPS (précision de l'ordre du mètre)
- Triangulation 3d (par sphères) 4 balises
  - *Essai déjà fait avec des émetteurs récepteurs RF pour le temps de vol (précision proportionnelle à la précision de l'horloge du système)*
  - *Mesure de l'amplitude du signal pour en déduire la distance*

# Modélisation des pièces :



Système de largage des graines



```

1 import numpy as np
2 import random
3 from pyfirmata import ArduinoMega
4 import time
5 import serial
6 from mpl_toolkits.mplot3d import Axes3D
7 import matplotlib.pyplot as plt
8 import matplotlib as cm
9
10
11
12
13 board=ArduinoMega('/dev/cu.usbmodem14101')
14 sensor= serial.Serial('/dev/cu.usbmodem14201', 9600)
15
16 sensx=board.get_pin('d:12:o')
17 motx=board.get_pin('d:3:p')
18 sensy=board.get_pin('d:13:o')
19 moty=board.get_pin('d:11:p')
20 ##Résolution
21 L=8
22 ##Résolution
23
24 ## Données initiales
25 angledroite=1 #a definir
26 anglemontee=2 #a definir
27 Carte=np.zeros((L,L))
28 p=0.4 #pas de vis A MODIFER
29 sens=1
30 ytemp=0.
31 commande=[0,1]
32 ligne=False
33 colonne=True
34 pos=[0,0]
35 vy=0.3 #vitesse du drone selon y
36 vx=0.5 #vitesse du drone selon x
37 dt=0.3 #1 microseconde correspond au temps d'échantillon
38 Vitessex= 1.5 #m.s-1
39 Vitessey= 2 #m.s-1
40 ## Données initiales

```

## Annexe 1 :

```

61 def retourCaseDepart():
62     global pos
63     global commande
64     global sens
65     global ytemp
66     global ligne
67     global colonne
68     (x,y)=pos
69     ##
70     part=min((x+1)/Vitessex,(y+1)/Vitessey)
71     sensx.write(1)
72     sensy.write(1)
73     motx.write(vx)
74     moty.write(vy)
75     time.sleep(part)
76     if part==(x+1)/Vitessex:
77         motx.write(0)
78         time.sleep((y)/Vitessey-part)
79         moty.write(0)
80     else:
81         moty.write(0)
82         time.sleep((x+1)/Vitessex-part)
83         motx.write(0)
84     ##
85     sens=1
86     ytemp=0.
87     commande=[0,1]
88     ligne=False
89     colonne=True
90     pos=[0,0]
91     return pos
92
93 def Mesure():
94     sensor.flushInput()
95     while sensor.readline()=='':
96         Mesure()
97     distance = sensor.readline()
98     return 12.1-float(distance)
99
100
101
102
103 def Mesure():
104     sensor.flushInput()
105     while sensor.readline()=='':
106         Mesure()
107     distance = sensor.readline()
108     return 12.1-float(distance)
109
110 def movex(): #fait déplacer le drone
111     global Carte
112     global pos
113     global X
114     global Y
115     global Z
116
117     #pos=getPos()
118     (x,y)=pos
119     if (int(x)<L) :
120         sensx.write(0)
121         motx.write(vx)
122         time.sleep(dt)
123         motx.write(0)
124         pos[0]+=dt*Vitessex*vx
125     z=Mesure()
126     X.append(x);Y.append(y);Z.append(z)
127
128 def movey(): #fait déplacer le drone
129     global Carte
130     global pos
131     global sens
132     global ytemp
133     global X
134     global Y
135     global Z
136
137     #pos=getPos()
138     (x,y)=pos
139     ytemp=y
140     if (0<=y<L-1) and sens==1:
141         sensy.write(0)
142         moty.write(vy)
143         time.sleep(dt)
144         moty.write(0)
145         pos[1]+=-dt*Vitessey*vy
146     if (1<=y<L) and sens== -1:
147         sensy.write(1)
148         moty.write(vy)
149         time.sleep(dt)
150         moty.write(0)
151         pos[1]+=-dt*Vitessey*vy
152
153 z=Mesure()
154 X.append(x);Y.append(y);Z.append(z)
155
156
157
158
159
160
161
162

```

# Annexe 2 : pythToArduino.py

```

177 def mouvementligne(): #fait changer la commande en accord
178
179     global pos
180     global sens
181     global ytemp
182
183     (x,y)=pos
184     if ytemp!=y and x<L-1 and (int(y)==0 or int(y)==L-1):
185         commande[0]+=1
186         while (not estDansCase()): #tant que le drone n'e
187             #anglex=0      #a utiliser sur le drone réel
188             #angley=angledroite    #a utiliser sur le dro
189             pos=movex()
190         ytemp=y #casse la boucle pour que mouvementligne
191         sens=-sens #on inverse le sens
192         commande[1]+=sens
193         print ('commande' + str(commande))

```

```

195 def mouvementcolonne():
196
197     global pos
198     global sens
199     global commande
200
201     #angley=0      #a utiliser sur le drone
202     y=pos[1]
203     while (not estDansCase()) and 0<=y<L:
204         pos=movey()
205         #anglex=sens*anglemontee #a utilis
206         if 1<=commande[1]<=L-2:
207             commande[1]+=sens

```

```

168 def estDansCase(): #Verifie que la co
169
170     global pos
171     global commande
172
173     (x,y)=pos;(xc,yc)=commande
174

```

```

222 def sondage():
223
224     global X
225     global Y
226     global Z
227     global Carte
228
229     for i in range (L*L-1):
230         mouvementcolonne()
231         mouvementligne()
232         retourCaseDepart()
233         M=sum(Z)/len(Z)
234         maxZ=max(Z)
235         sol=np.array([[0,1,0],[1,0,0]])
236         sol=ListedColormap(sol)
237         fig=plt.figure()
238         ax=fig.add_subplot(projection='3d')
239         surf=ax.plot_trisurf(X,Y,Z,cmap = sol,vmin=M-maxZ,vmax=M+maxZ)
240         bar=fig.colorbar(surf, shrink=0.5, aspect=5)
241         bar.set_label("Limite de plantation : " +str("%.2f" % M))
242         plt.show()
243         setCarte(X,Y,Z,M)

```

```

213 def setCarte(x,y,z,m):
214     global Carte
215     n=len(x)
216     for i in range (n):
217         if z[i]>m:
218             Carte[int(x[i]),int(y[i])]=0
219         else:
220             Carte[int(x[i]),int(y[i])]=1

```

# Annexe 3 : PIDx.py

```
1 Kp=1
2 Kd=1
3 Ki=1
4 def movex(C,posPrec,err):#C est une liste [xc,yc], err=sum(xc-x)
5     global Carte
6     global pos
7     global X
8     global Y
9     global Z
10    [xc,yc]=posPrec
11    pos=getPos() #renvoyé par le capteur de position
12    (x,y)=pos
13    err+=(xc-x)
14    if xc>x: #le moteur avance
15        sensx.write(0)
16        motx.write(min(abs(vx*(Kp*(xc-x)+Kd*(xc-x)/dt+Ki*err)),1))
17    if xc<x: #le moteur recule
18        sensx.write(1)
19        motx.write(min(abs(vx*(Kp*(xc-x)+Kd*(xc-x)/dt+Ki*err)),1))
20    else:
21        pass
22    Carte[int(x),int(y)]=estPlantable()
23    z=Mesure()
24    X.append(x);Y.append(y);Z.append(z)
25    return pos
```