Analyse dynamique d'une caméra câblée

Etudier la relation entre la trajectoire d'une caméra câblée et son système de support et de commande. Déterminer les paramètres influant sur le système afin d'assurer les performances souhaitées.

N° d'inscription : 10695

PLAN

<u>Etude théorique et simulation</u>

- Présentation et modélisation
- Analyse dynamique
- Simulation et résultats

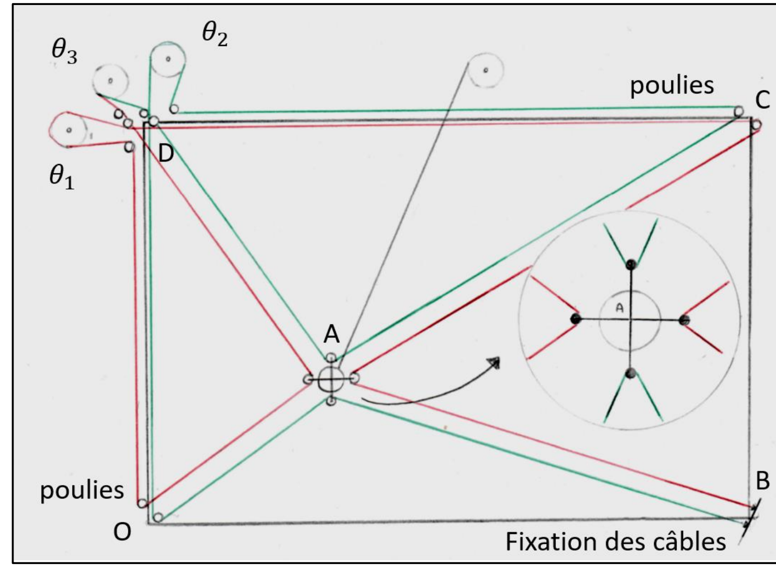
<u>Expérimentation</u>

- Elaboration du cahier des charges
- Présentation, objectifs et protocole
- Comparaison aux résultats théoriques

Présentation et modélisation

Hypothèses simplificatrices

- Caméra ponctuelle
- Poulies de rayon négligeable par rapport à la dimension du terrain
- Câbles supposés inextensibles et tendus.



Vue de dessus

Modélisation des câbles

$$k = \sqrt{x^2 + y^2 + z^2}$$
$$h = \sqrt{(d - x)^2 + y^2 + z^2}$$
$$m = \sqrt{(d - x)^2 + (s - y)^2 + z^2}$$
$$n = \sqrt{x^2 + (s - y)^2 + z^2}$$

Modélisation des moteurs

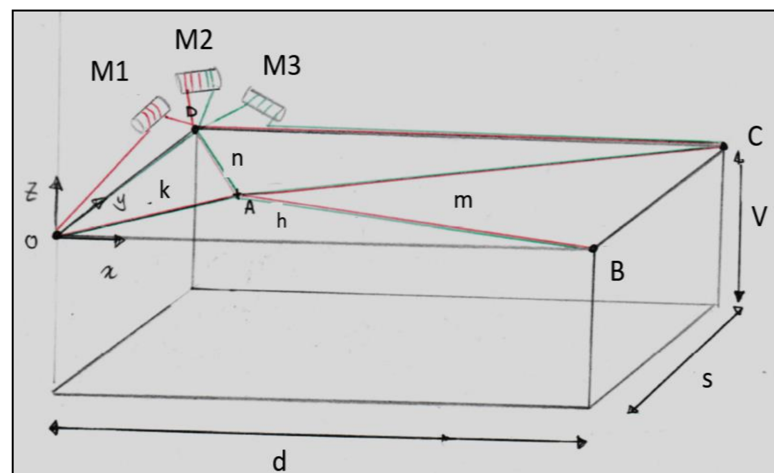
$$R\dot{\theta}_1 = \dot{m} + \dot{h}$$
$$R\dot{\theta}_2 = \dot{k} + \dot{h}$$
$$R\dot{\theta}_3 = \dot{m} + \dot{n} + \dot{k} + \dot{h}$$

(R : rayon de chaque moteur)

Relation entre la vitesse de rotation des moteurs et la vitesse de la caméra

On note X = $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ et H = $\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$: $\boxed{\dot{H} = P\dot{X}}$

$$P = \frac{1}{R}\begin{bmatrix} \frac{-(d-x)}{m} + \frac{-(d-x)}{h} & \frac{-(s-y)}{m} + \frac{y}{h} & \frac{z}{m} + \frac{z}{h} \\ \frac{x}{k} + \frac{-(d-x)}{h} & \frac{y}{k} + \frac{y}{h} & \frac{z}{k} + \frac{z}{h} \\ \frac{-(d-x)}{m} + \frac{x}{n} + \frac{x}{k} + \frac{-(d-x)}{h} & \frac{-(s-y)}{m} + \frac{-(s-y)}{n} + \frac{y}{k} + \frac{y}{h} & \frac{z}{m} + \frac{z}{n} + \frac{z}{k} + \frac{z}{h} \end{bmatrix}$$



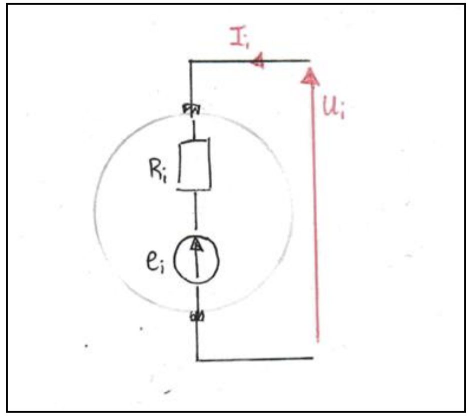
Modélisation du système en 3 dimensions

Modélisation des moteurs

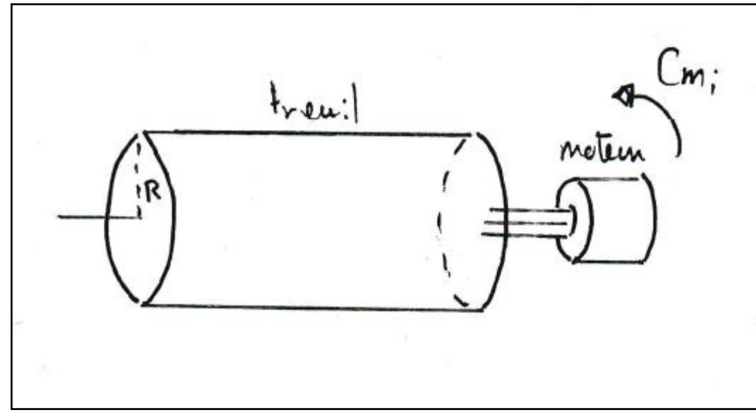
Pour i∈ ⟦1,3⟧,

Caractéristiques MCC
Ri : résistance d’induit
Ei : fem à vide
Cm : couple moteur
Ji : moment d’inertie

On note U = $\begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix}$ et F = $\begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}$



Modélisation électrique



Modélisation mécanique

Equation électrique

$$Cm_i = Ki_i(U_i - Ke_i\dot{\theta}_i)/R_i$$

Equation mécanique

$$J_i\ddot{\theta}_i = Cm_i - Cr_i - \lambda_i\dot{\theta}_i$$

$$U = \begin{bmatrix} \frac{R_1J_1}{Ki_1} & 0 & 0 \\ 0 & \frac{R_2J_2}{Ki_2} & 0 \\ 0 & 0 & \frac{R_3J_3}{Ki_3} \end{bmatrix}\ddot{H} + \begin{bmatrix} Ke_1 + \frac{\lambda_1J_1}{Ki_1} & 0 & 0 \\ 0 & Ke_2 + \frac{\lambda_2J_2}{Ki_2} & 0 \\ 0 & 0 & Ke_3 + \frac{\lambda_3J_3}{Ki_3} \end{bmatrix}\dot{H} + R\begin{bmatrix} \frac{R_1}{Ki_1} & 0 & 0 \\ 0 & \frac{R_2}{Ki_2} & 0 \\ 0 & 0 & \frac{R_3}{Ki_3} \end{bmatrix}F$$

$$\boxed{\boldsymbol{U = G\ddot{H} + L\dot{H} + RSF}}$$

Equation de mouvement

Force de tension résultante des câbles

$$\vec{T} = (T_k^r + T_k^v)\frac{\overrightarrow{AO}}{k} + (T_n^r + T_n^v)\frac{\overrightarrow{AD}}{n} + (T_m^r + T_m^v)\frac{\overrightarrow{AC}}{m} + (T_h^r + T_h^v)\frac{\overrightarrow{AB}}{h}$$

Relation avec F :

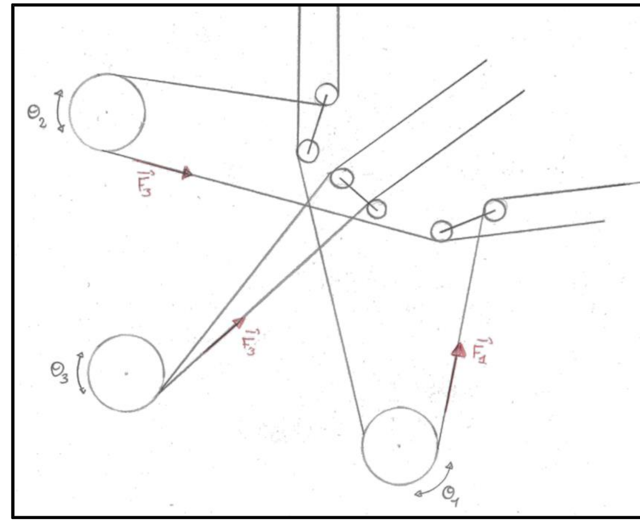
$$F_1 = +T_k^r - T_m^r \ , \qquad F_2 = +T_m^v - T_k^v \ , \qquad F_3 = +T_m^v - T_m^r$$

Conservation de la tension au point d'accroche :

$$T_k^v = T_h^v \ , \qquad T_k^r = T_n^r \ , \quad T_n^b = T_m^v \ , \qquad T_h^r = T_m^r$$

Répartition équitable au niveau du troisième moteur :

$$T_n^v = T_n^r$$



Forces de tension au niveau des 3 moteurs

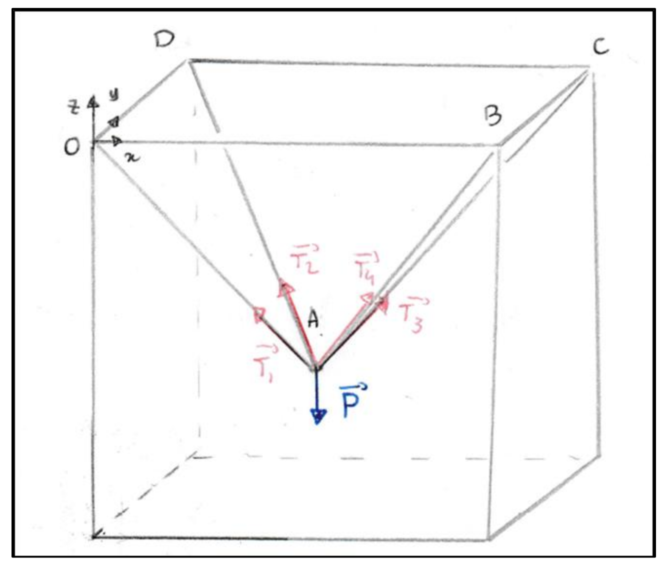
Relation entre les forces de tension au sein des moteurs et celles au sein de la caméra

$$\text{On note T} = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \text{ et F} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}, \boxed{T = RP^T F}$$

Principe fondamental de la dynamique

$$m_A \vec{A} = \vec{P} + \vec{T} + \overrightarrow{F_{frott}} + \overrightarrow{F_{pert}}$$

$$F = R(P^T)^{-1}(m_A \ddot{X} + P - F_{pert,tot})$$



Résultante des forces au niveau de la caméra

Résolution numérique

Relation entre la rotation des moteur et la position de la caméra

$$\dot{H} = P\dot{X}$$

Equation de mouvement

$$U = G\ddot{H} + L\dot{H} + RSF$$

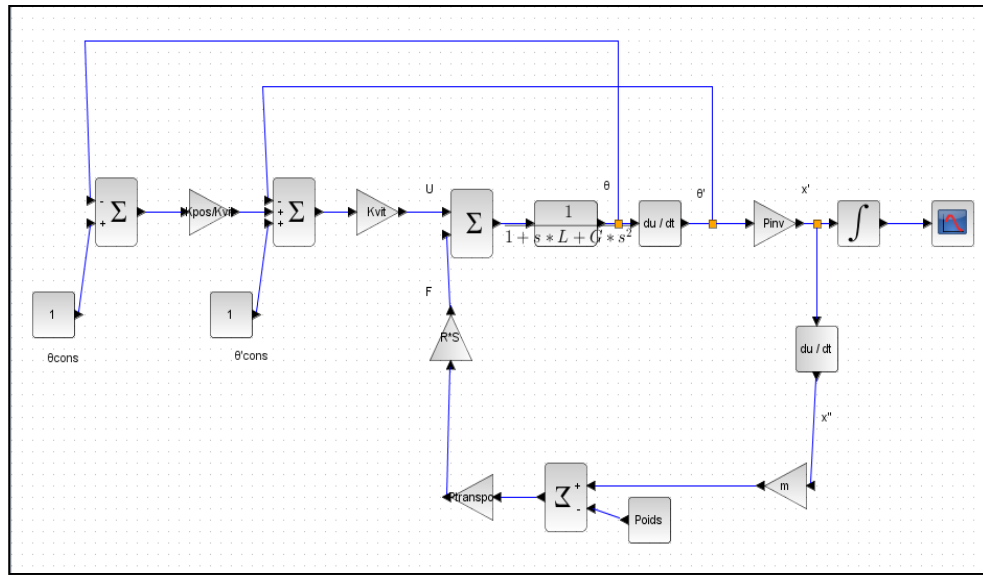
Avec $\ddot{H} = P\ddot{X} + \dot{P}\dot{X} = P\ddot{X} + \begin{bmatrix} t_{x1}Z_1\dot{X} \\ t_{x2}Z_2\dot{X} \\ t_{x3}Z_3\dot{X} \end{bmatrix}$

Expression de la force résultant du PFD

$$F = R(P^T)^{-1}(m_A\ddot{X} + P - F_{pert,tot})$$

Loi de commande

$$U = K_{vit}(\dot{H} - \dot{H}^{\circ}) + K_{pos}(H - H^{\circ})$$

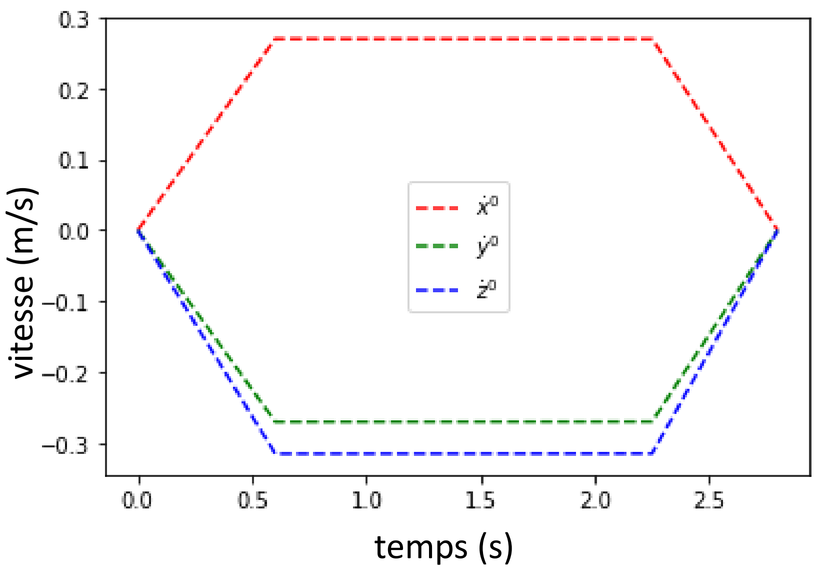


Commande simulation

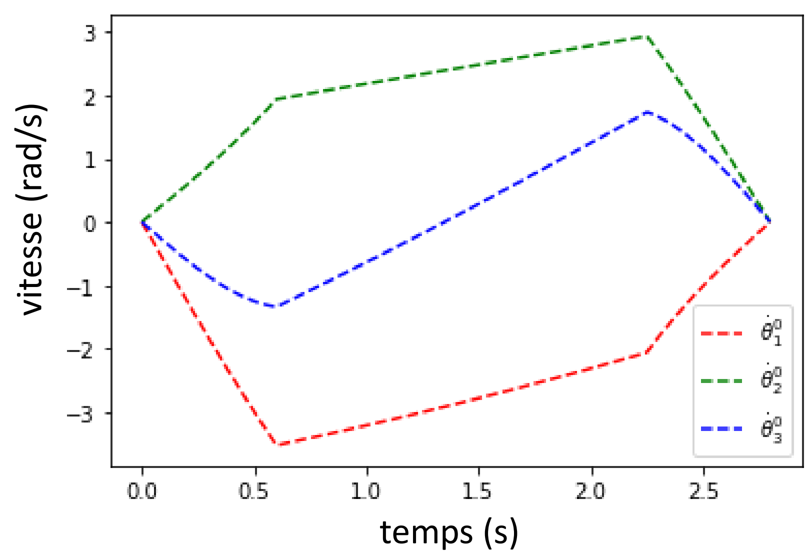
Données numériques

```
g=9.81                                  # Accélération de la pesanteur
mA=1                                    # Masse de la caméra et du support
d=3.2; s=2.2; v=2.0; STADE=[d,s,v]      # Dimension du stade
R=0.15                                  # Rayon des treuils
R1=0.917;    R2=0.917;    R3=0.917      # Résistance d'induit des moteurs
J1=1.5859;   J2=1.5859;   J3=1.5859;    # Moment d'inertie des treuils
kI1=2.51945; kI2=2.51945; kI3=2.51945;  # Constante liant couple et courant des moteurs
kE1=3.3942;  kE2=3.3942;  kE3=3.3942;   # Constante liant vitesse et fem des moteurs
la1=0.0670;  la2=0.0670;  la3=0.0670;   # Coefficient de frottement des moteurs
Kpos=4200;   Kvit=130;                  # Gains des correcteurs
```

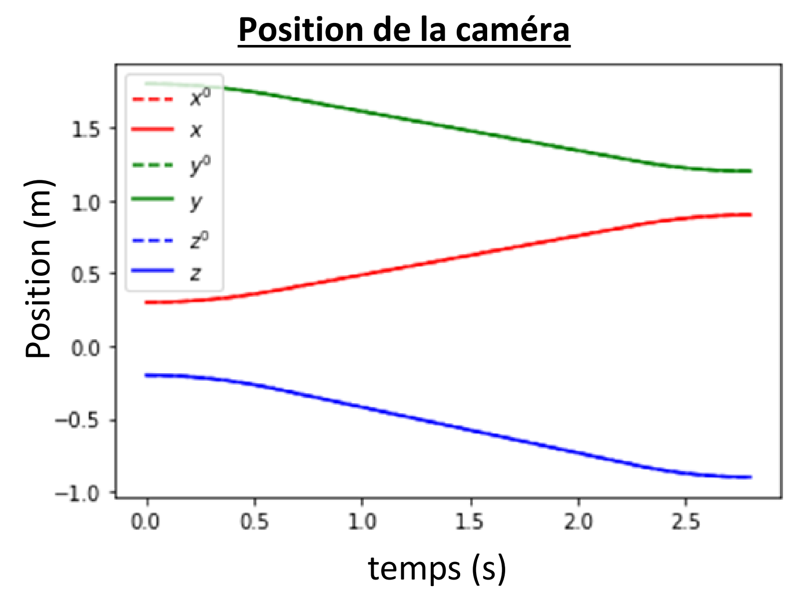
Vitesse de référence

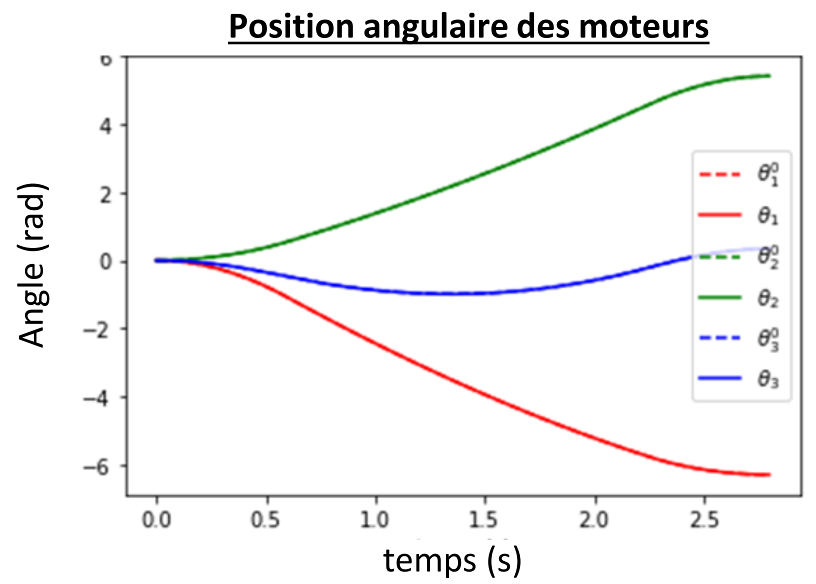


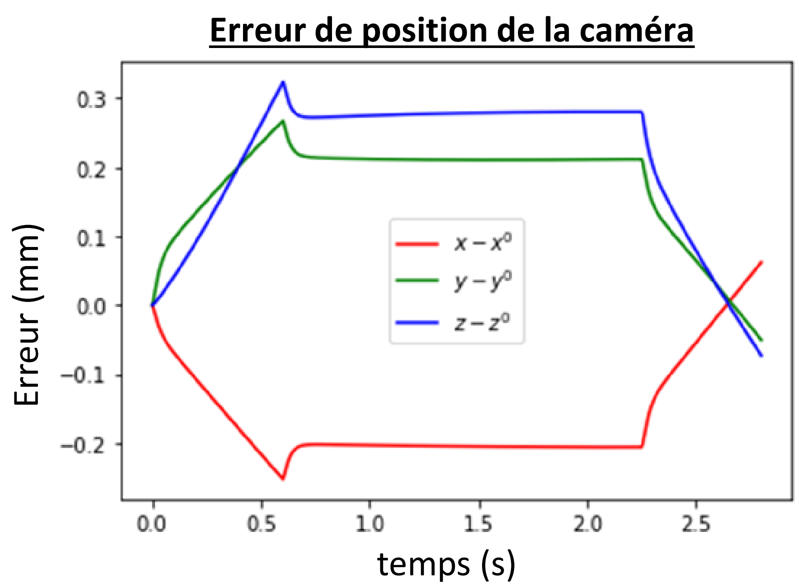
Vitesse de rotation de référence des moteurs

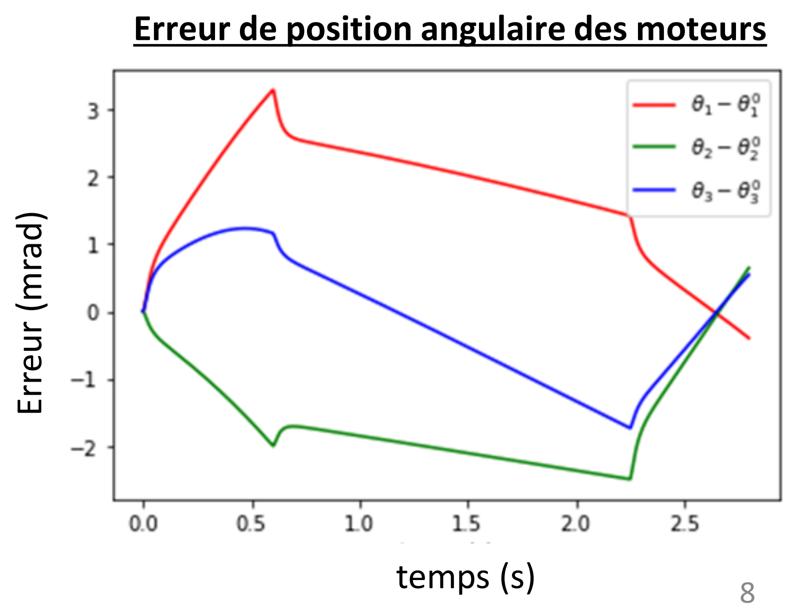


Résultats simulation



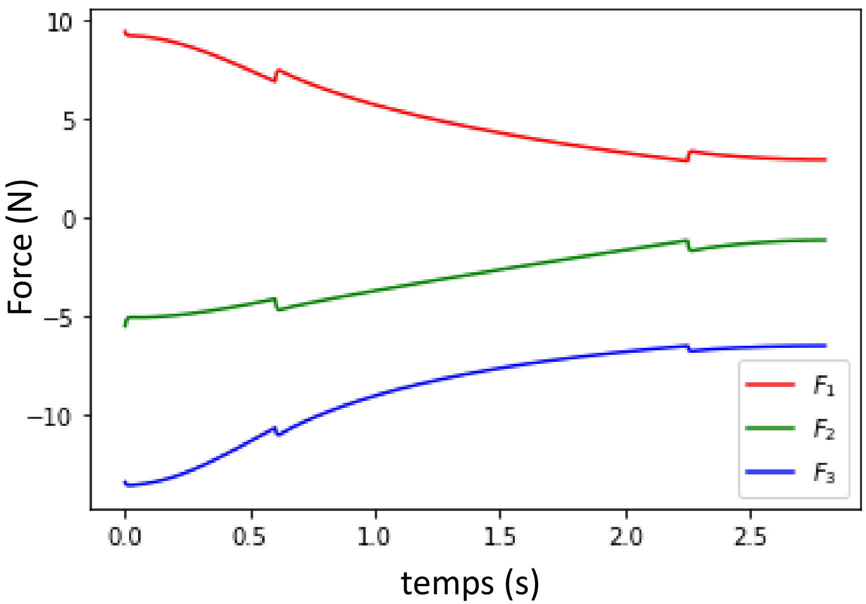




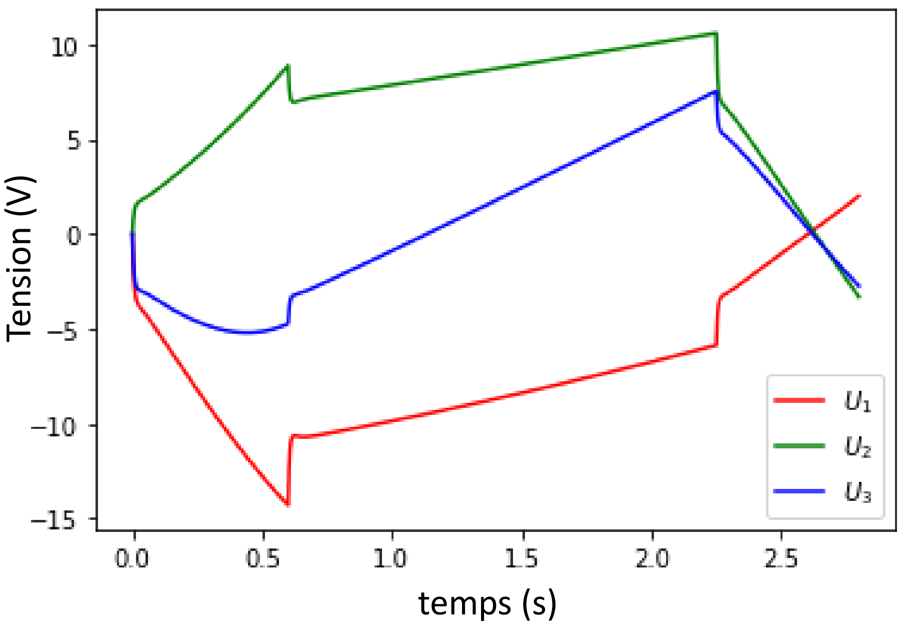


Résultats simulation

Force de tension au niveau des 3 treuils



Tension de commande des moteurs



Présentation du modèle expérimental

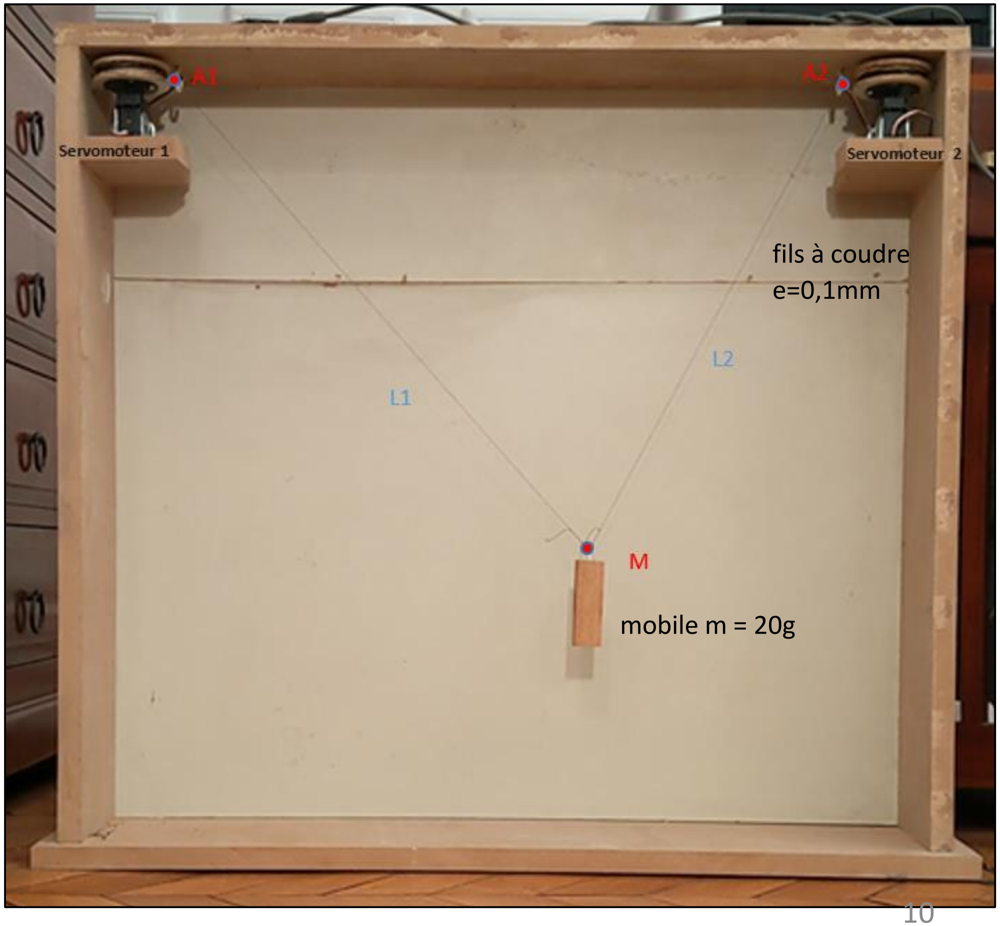
Objectifs :

- Réaliser une trajectoire prédéfinie
- Confirmer les relations du PGD, PGI

Servomoteur avec poulie (rayon r = 2,8 cm)

Carte Arduino avec moteur shield

Banc d'essai (56x64cm): système de poulie avec servomoteurs

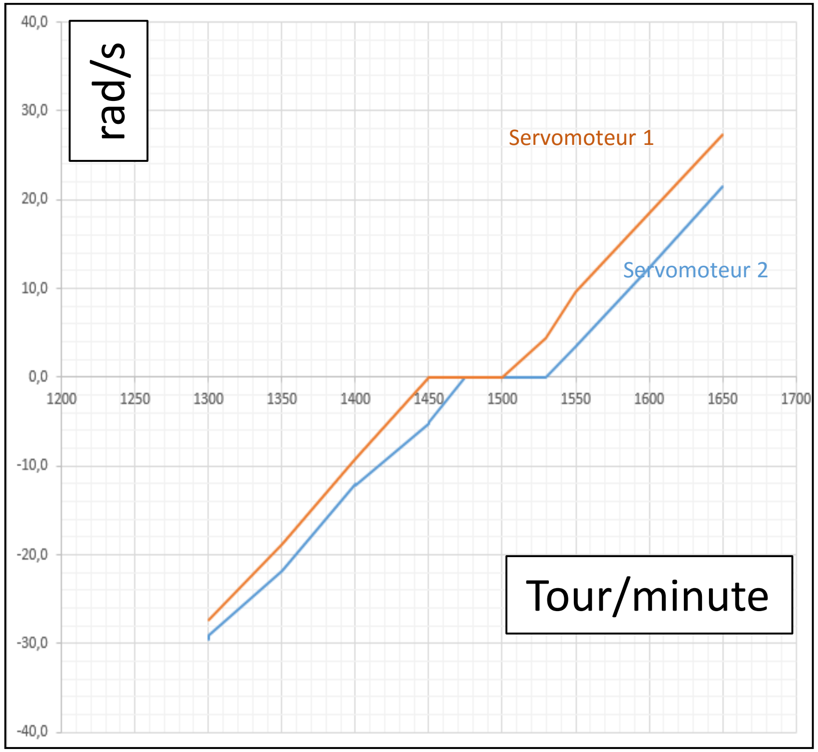


Cahier des charges

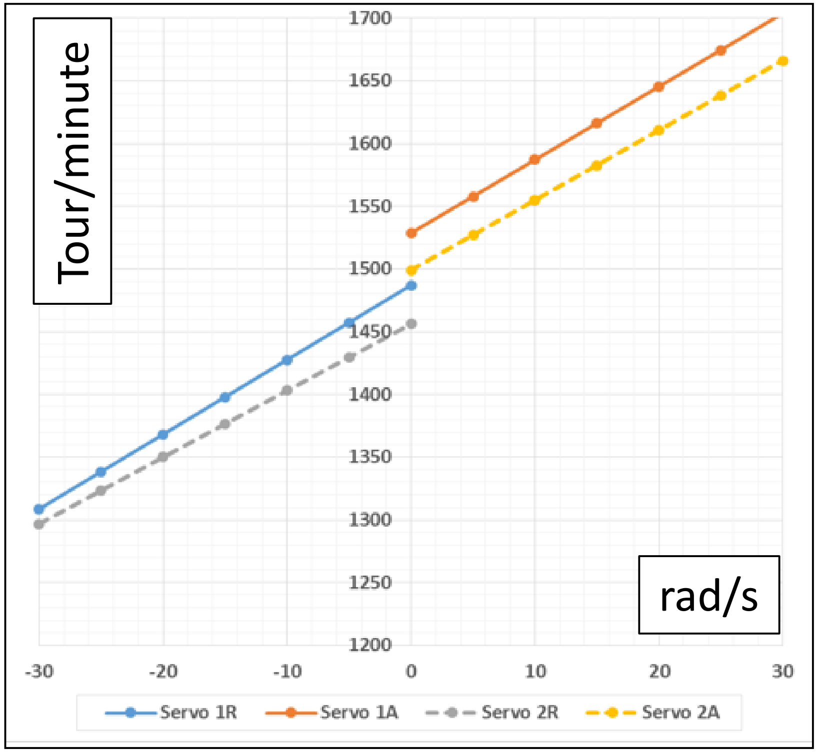
Exigences	Contraintes	Niveau
Se déplacer dans un espace prédéfini	Plan de travail Perte de hauteur Tension maximale dans chaque câble	[0.64 m, 0.56 m] <10% 5 N
Architecture du robot	Masse du robot soutenable	> 0,100 kg
Avoir une vitesse importante	Vitesse maximale caméra	>0.05 m/s
Contrôle du robot	Respecte la position Respecte la vitesse	+-10% max de position consigne +-10% max de vitesse consigne
Ne pas déranger les téléspectateurs	Finesse câble	<1mm

Loi de vitesse : commande des servomoteurs

Mesure de vitesse pour une commande fixée

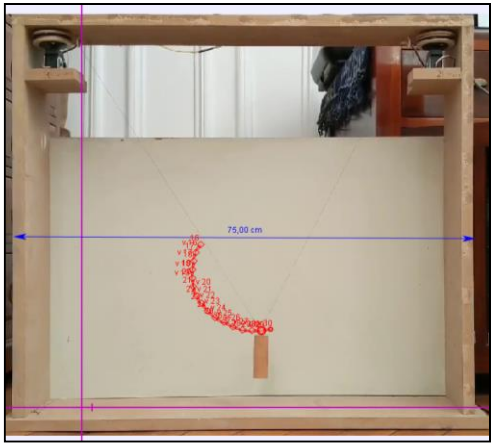


Relation linéaire inversée pour la commande

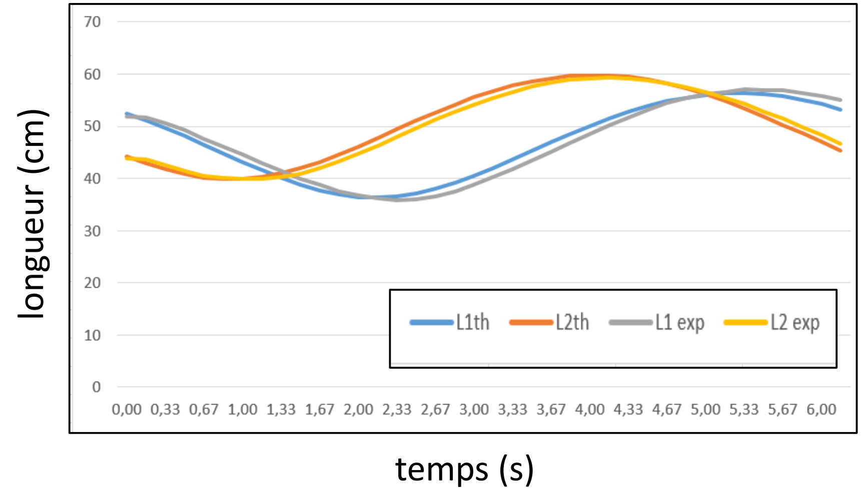


Comparaison avec la commande

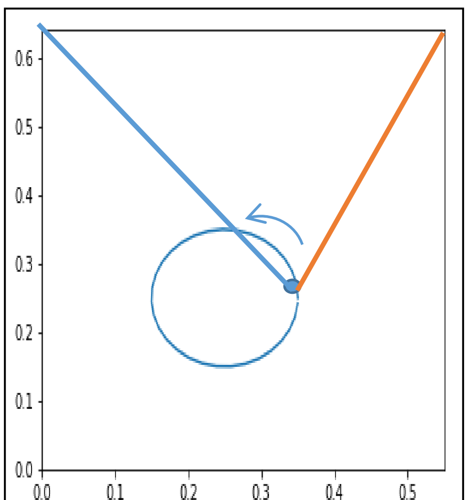
Trajectoire étudiée



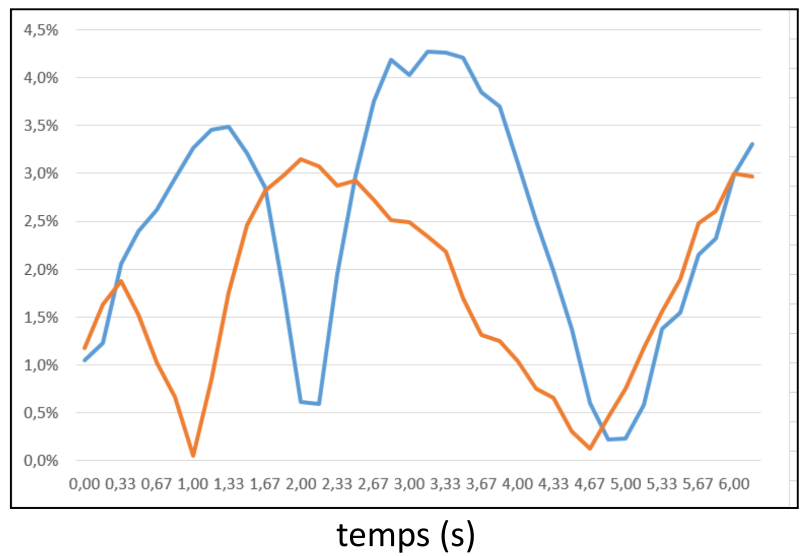
Evolution de la longueur des câbles



Modélisation trajectoire

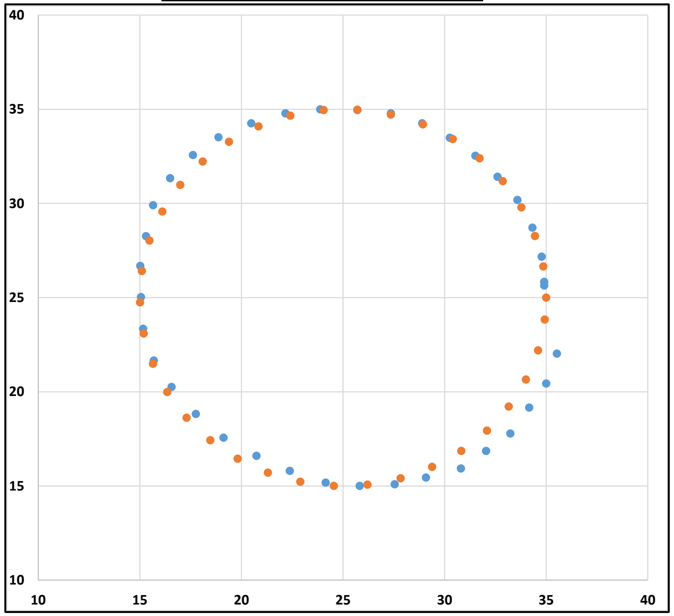


Erreur sur L1 et L2 (%)



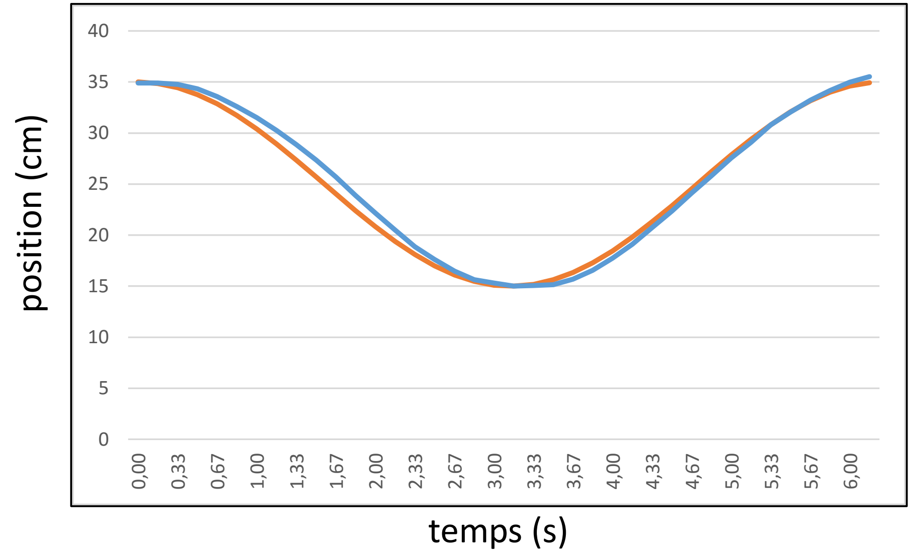
Réponse du module a la commande souhaitée

Evolution du module

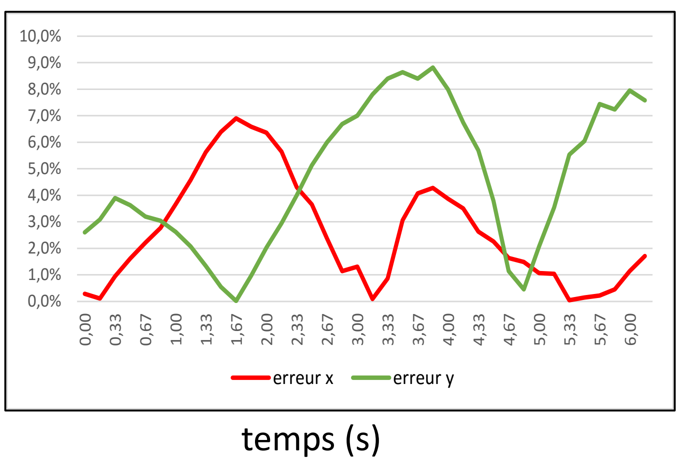


Trajectoire circulaire théorique (orange) et mesurées (bleu)

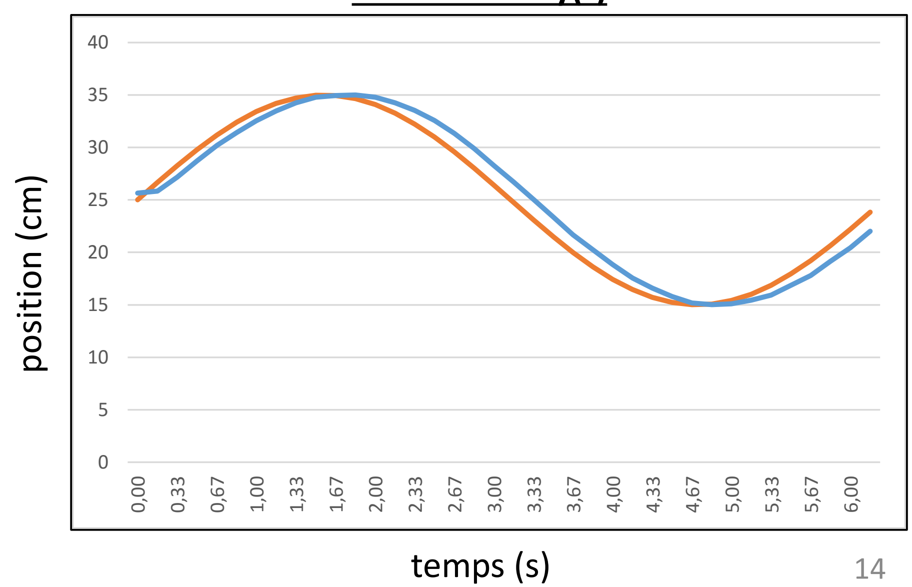
Evolution de x(t)



Erreur position (%)

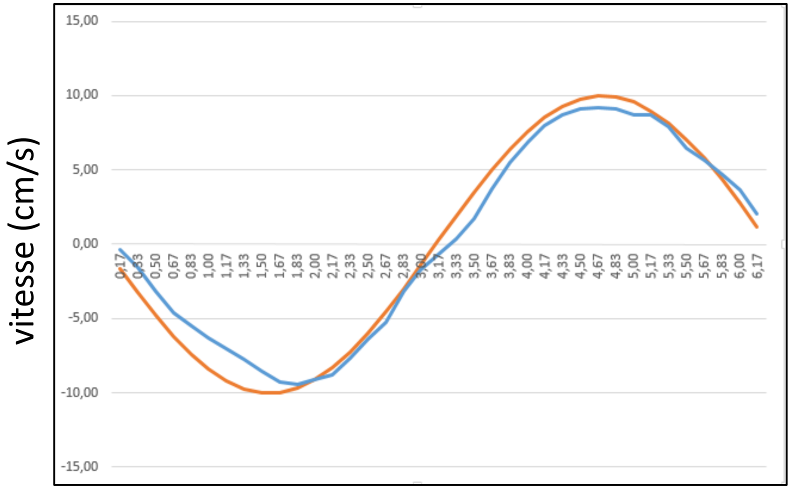


Evolution de y(t)

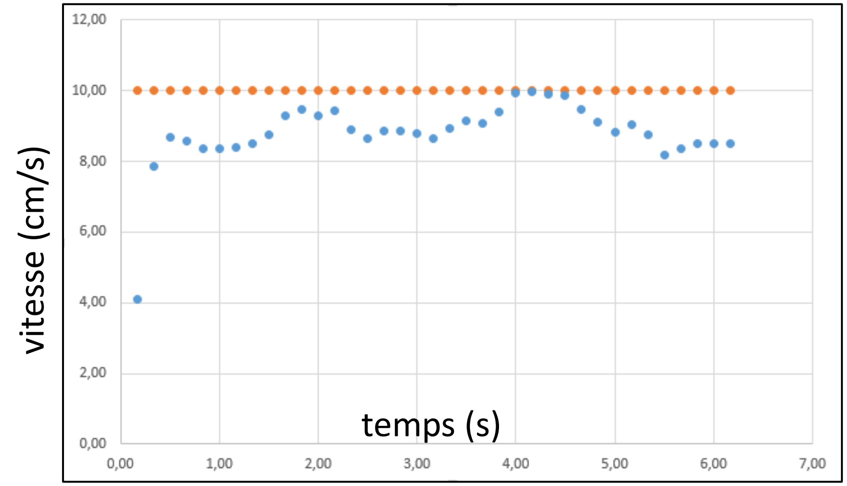


Etude de la vitesse du module

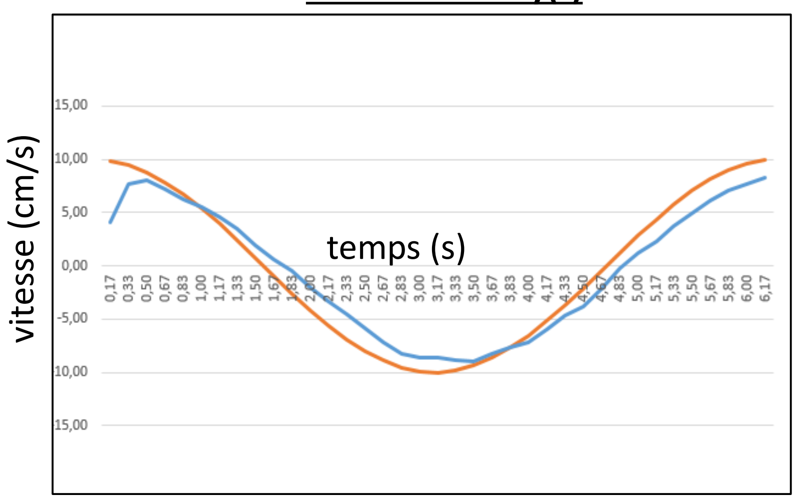
Evolution de Vx(t)



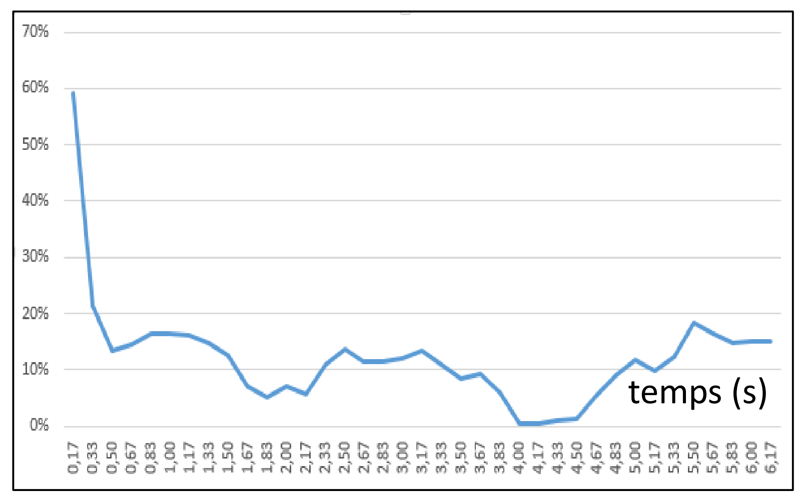
Evolution de V(t) (vitesse absolue)



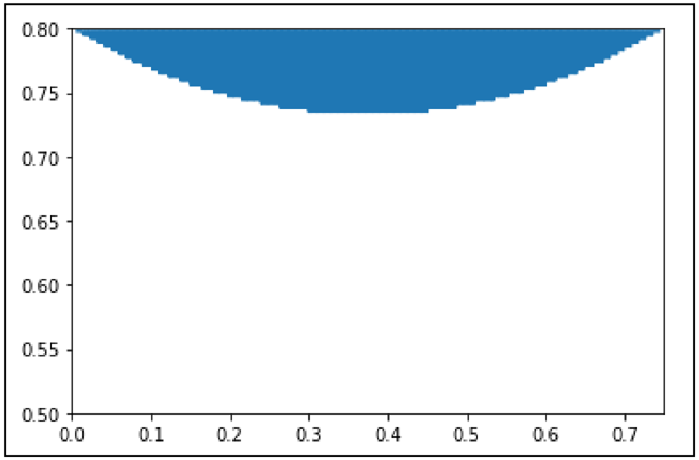
Evolution de Vy(t)



Erreur de vitesse (%)



Validation de l'espace de travail



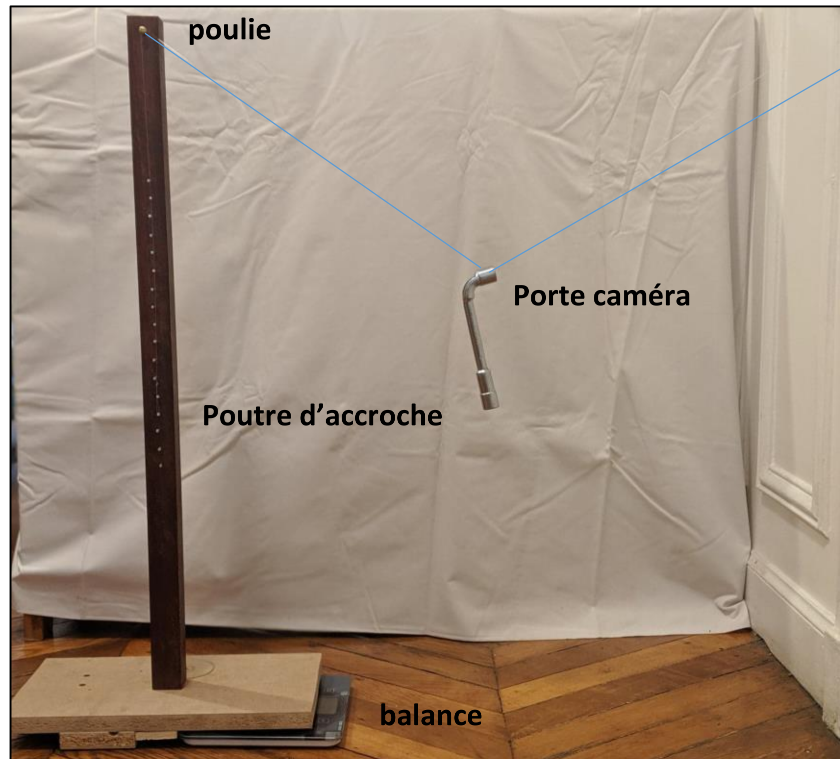
Espace de travail théorique pour masse m = 0,180 kg

ord./abs.	0,1	0,2	0,3	0,4	0,5	0,6	0,7
0,8							
0,77							
0,75							
0,72							
0,7							
0,67							
0,65							
0,62							
0,6							
0,57							

Espace de travail expérimental pour masse m = 0,180 kg

PFD appliqué au porte caméra

$$\begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \frac{1}{\sin(\theta_1 + \theta_2)} \begin{bmatrix} -sin\theta_2 & cos\theta_2 \\ sin\theta_1 & cos\theta_1 \end{bmatrix} \begin{bmatrix} 0 \\ mg \end{bmatrix}$$



Banc d'essai pour la mesure de force de tension

Conclusion

```python
from math import *
import numpy as np
import scipy.integrate as spint


"Construction de la matrice de changement de Base Cart.->Theta --"

def CalcPXTH(X,STADE,R):
    """
    Calcul de la matrice P
    entrée :    X position (vecteur 3 composantes)
                STADE taille du stade (vecteur 3 composantes)
                R rayon des treuils (scalaire)
    sortie : matrice 3x3
    """
    x=X[0]; y=X[1]; z=X[2]
    d=STADE[0]; s=STADE[1]; v=STADE[2]
    k=np.sqrt(x**2+y**2+z**2)
    h=np.sqrt((d-x)**2+y**2+z**2)
    m=np.sqrt((d-x)**2+(s-y)**2+z**2)
    n=np.sqrt(x**2+(s-y)**2+z**2)

    P=np.zeros((3,3))
    P[0,0]=-(d-x)/m-(d-x)/h;         P[0,1]=-(s-y)/m+y/h;             P[0,2]=z/m+z/h;
    P[1,0]=-(d-x)/m+x/n    ;         P[1,1]=-(s-y)/m-(s-y)/n;         P[1,2]=z/m+z/n;
    P[2,0]=-(d-x)/m+x/n+x/k-(d-x)/h; P[2,1]=-(s-y)/m-(s-y)/n+y/k+y/h; P[2,2]=z/m+z/n+z/k+z/h;

    return P/R

```

```python
"Construction des  matrices H1, H2, H3 de changement de Base Cart.->Theta v2.0-"

def CalcHXTH(X,STADE,R):
    """
    Calcul des matrices H1, H2, H3
    entrée :    X position (vecteur 3 composantes)
                STADE taille du stade (vecteur 3 composantes)
                R rayon des treuils (scalaire)
    sortie : H1,H2,H3 matrices 3x3
    """
    x=X[0]; y=X[1]; z=X[2]
    d=STADE[0]; s=STADE[1]; v=STADE[0]
    k=np.sqrt(x**2+y**2+z**2)
    h=np.sqrt((d-x)**2+y**2+z**2)
    m=np.sqrt((d-x)**2+(s-y)**2+z**2)
    n=np.sqrt(x**2+(s-y)**2+z**2)

    H1=np.zeros((3,3))
    H1[0,0]=1/m+1/h-((d-x)**2)/m**3-((d-x)**2)/h**3;    H1[0,1]=-(d-x)*(s-y)/m**3+(d-x)*y/h**3;         H1[0,2]=(d-x)*z/m**3+(d-x)*z/h**3;
    H1[1,0]=-(d-x)*(s-y)/m**3+(d-x)*y/h**3;             H1[1,1]=1/m+1/h-((s-y)**2)/m**3-(y**2)/h**3;     H1[1,2]=(s-y)*z/m**3-y*z/h**3;
    H1[2,0]=(d-x)*z/m**3+(d-x)*z/h**3;                  H1[2,1]=(s-y)*z/m**3-y*z/h**3;                   H1[2,2]=1/m+1/h-(z**2)/m**3-(z**2)/h**3;

    H2=np.zeros((3,3))
    H2[0,0]=1/m+1/n-((d-x)**2)/m**3-(x**2)/n**3;        H2[0,1]=-(d-x)*(s-y)/m**3+x*(s-y)/n**3;          H2[0,2]=(d-x)*z/m**3-x*z/n**3;
    H2[1,0]=-(d-x)*(s-y)/m**3+x*(s-y)/n**3;             H2[1,1]=1/m+1/n-((s-y)**2)/m**3-((s-y)**2)/n**3; H2[1,2]=(s-y)*z/m**3+(s-y)*z/n**3;
    H2[2,0]=(d-x)*z/m**3-x*z/n**3;                      H2[2,1]=(s-y)*z/m**3+(s-y)*z/n**3;               H2[2,2]=1/m+1/n-(z**2)/m**3-(z**2)/n**3;

    H3=np.zeros((3,3))
    H3[0,0]=1/m+1/n+1/k+1/h-((d-x)**2)/m**3-(x**2)/n**3-(x**2)/k**3-((d-x)**2)/h**3;    H3[0,1]=-(d-x)*(s-y)/m**3+x*(s-y)/n**3-x*y/k**3+(d-x)*y/h**3;                H3[0,2]=(d-x)*z/m**3-x*z/n**3-x*z/k**3+(d
    H3[1,0]=-(d-x)*(s-y)/m**3+x*(s-y)/n**3-x*y/k**3+(d-x)*y/h**3;                       H3[1,1]=1/m+1/n+1/k+1/h-((s-y)**2)/m**3-((s-y)**2)/n**3-(y**2)/k**3-(y**2)/h**3;  H3[1,2]=(s-y)*z/m**3+(s-y)*z/n**3-y*z/k**
    H3[2,0]=(d-x)*z/m**3-x*z/n**3-x*z/k**3+(d-x)*z/h**3;                               H3[2,1]=(s-y)*z/m**3+(s-y)*z/n**3-y*z/k**3-y*z/h**3;                          H3[2,2]=1/m+1/n+1/k+1/h-(z**2)/m**3-(z**2

    return H1/R,H2/R,H3/R
```

```python

"Détermination du trajet de référence en fonction du temps"


def TrajRef(Xdep,Xfin,tfin,t1,t2,STADE,R,timestep=0.001):
    """
    Calcul de la trajectoire de référence
    entrée :    Xdep position de départ (vecteur colonne 3 composantes)
                Xfin position d'arrivée (vecteur colonne 3 composantes)
                tfin date d'arrivée (scalaire)
                t1 date à la fin de la première accélération (scalaire)
                t2 date au début du freinage (scalaire)
                STADE taille du stade (vecteur 3 composantes)
                R rayon des treuils (scalaire)
                dt temps d'échantillonnage (scalaire - optionnel)
    sortie : temps t (vecteur ligne taille N)
             vitXref position de référence (matrice 3 lignes N colonnes)
             posXref vitesse de référence (matrice 3 lignes N colonnes)
             vitThref vitesse angulaire de référence (matrice 3 lignes N colonnes)
             posThref position angulaire de référence (matrice 3 lignes N colonnes)
    """
    Nt=int((tfin-0)/timestep+1)
    t,dt=np.linspace(0,tfin,Nt,retstep=True)

    V0=2*(Xfin-Xdep)/(tfin+(t2-t1))
    vitXref=(t<=t1)          *(V0*t/t1)+\
        ((t>t1)*(t<=t2))*V0       +\
        (t>t2)          *(-V0/(tfin-t2)*(t-t2)+V0)
    posXref=(t<=t1)          *(V0*t**2/(2*t1)+Xdep)+\
        ((t>t1)*(t<=t2))*(V0*(t-t1)+V0*t1/2+Xdep)+\
        (t>t2)          *(-V0/(2*(tfin-t2))*(t-t2)**2+V0*(t-t1)+V0*t1/2+Xdep)
    vitThref=np.zeros((3,Nt))
    for i in range(Nt):
        PXth=CalcPXTH(posXref[:,i],STADE,R)
        vitThref[:,i]=np.dot(PXth,vitXref[:,i])
        posThref=np.zeros((3,Nt))
    for k in range(3):
        posThref[k,:]=spint.cumtrapz(vitThref[k,:],initial=0)*dt

    return t,vitXref,posXref,vitThref,posThref
```

```python
def CalcConsigne(t,time,vitXref,posXref,vitThref,posThref):
    """"
    Formation des signaux de consigne pour la régulation par interpolation
    à partir du calcul de la trajectoire de référence
    entrée  : t date à laquelle on souhaite déterminer la consigne (scalaire)
              time temps (vecteur ligne taille N)
              vitXref position de référence (vecteur 3 lignes N colonnes)
              posXref vitesse de référence (vecteur 3 lignes N colonnes)
              vitThref vitesse angulaire de référence (vecteur 3 lignes N colonnes)
              posThref position angulaire de référence (vecteur 3 lignes N colonnes)
    sortie : X_pt0 vitesse de consigne (vecteur 3 colonnes)
             X0    position de consigne (vecteur 3 colonnes)
             Th_pt0 vitesse angulaire de consigne (vecteur 3 colonnes)
             Th0 position angulaire de consigne (vecteur 3 colonnes)
    """
    X_pt0=np.zeros((3,1))
    X0=np.zeros((3,1))
    Th0=np.zeros((3,1))
    Th_pt0=np.zeros((3,1))
    for k in range(3):
        X_pt0[k]=np.interp(t,time,vitXref[k,:])
        X0[k]=np.interp(t,time,posXref[k,:])
        Th_pt0[k]=np.interp(t,time,vitThref[k,:])
        Th0[k]=np.interp(t,time,posThref[k,:])
    return X_pt0,X0,Th_pt0,Th0
```

```python

"Défintion du système différentiel à résoudre"

def EqDiff(Y,t,STADE,R,mA,Sinv,L,G,g,Kpos,Kvit,time,vitXref,posXref,vitThref,posThref,Fpert):
    """
    Equation différentielle régissant le système  (Y)'=f(Y,t)
    entrée : Y vecteur 12 composantes
             t temps (scalaire)
             ... tous les arguments nécessaires au calcul
    sortie : Y' vecteur 12 composantes

    """
    Ypt=np.zeros(12)
    X   = Y[0:3,None]
    Xpt = Y[3:6,None]
    Th  = Y[6:9,None]
    Thpt= Y[9:12,None]

    P=CalcPXTH(X,STADE,R)
    H1,H2,H3=CalcHXTH(X,STADE,R)
    Xpt0,X0,Thpt0,Th0=CalcConsigne(t,time,vitXref,posXref,vitThref,posThref)

    M=np.dot(P.transpose(),Sinv)/mA
    N=np.linalg.inv(np.eye(3)+np.dot(np.dot(M,G),P))
    A=np.array([[np.dot(np.dot(Xpt.transpose(),H1),Xpt)[0][0]],[np.dot(np.dot(Xpt.transpose(),H2),Xpt)[0][0]],[np.dot(np.dot(Xpt.transpose(),H3),Xpt)[0][0]]])
    B=np.dot(L,Thpt)
    C=Kpos*(Th0-Th)
    D=Kvit*(Thpt0-Thpt)
    E=-np.array([[0],[0],[g]])+Fpert(t)/mA


    Ypt[0:3]=Xpt.transpose()
    Ypt[6:9]=Thpt.transpose()
    Xsec=np.dot(N,-np.dot(M,A)-np.dot(M,B)+np.dot(M,C)+np.dot(M,D)+E)
    Thsec=np.dot(P,Xsec)+A
    Ypt[3:6]=Xsec.transpose()
    Ypt[9:12]=Thsec.transpose()

    return Ypt
```

```python
from IPython import get_ipython
get_ipython().magic('clear')
get_ipython().magic('reset -sf')

from math import *
from cameralib import *
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as spint


"CONTEXTE"

dt=0.0001                                  # Période d'échantillonnage des signaux
g=9.81                                     # Accélération de la pesanteur
mA=1                                       # Masse de la caméra et du support
d=3.2; s=2.2; v=2.0; STADE=[d,s,v]         # Dimension du stade
R=0.15                                     # Rayon des treuils
R1=0.917;     R2=0.917;     R3=0.917       # Résistance d'induit des moteurs
J1=1.5859;    J2=1.5859;    J3=1.5859;     # Moment d'inertie des treuils
kI1=2.51945; kI2=2.51945; kI3=2.51945;     # Constante liant couple et courant des moteurs
kE1=3.3942;   kE2=3.3942;   kE3=3.3942;    # Constante liant vitesse et fem des moteurs
la1=0.0670;   la2=0.0670;   la3=0.0670;    # Coefficient de frottement des moteurs
Kpos=4200;    Kvit=130;                    # Gains des correcteurs
G=np.diag([R1*J1/kI1,R2*J2/kI2,R3*J3/kI3]) # Construction des matrices G, L et S
L=np.diag([kE1+la1*R1/kI1,kE2+la2*R2/kI2,kE3+la3*R3/kI3])
S=np.diag([R1/kI1,R2/kI2,R3/kI3]); Sinv=np.linalg.inv(S)

"Force de perturbation"

def Force_pert(t):

    Fp=np.array([[(100*(sin(4*pi*t)+sin(32*pi*t)))],[0],[0]])
    return Fp

```

```python
"Trajéctoire de référence"

tfin=2.8
t1=0.6; t2=2.25
Xdep=np.array([[0.3],[1.8],[-0.2]])
Xfin=np.array([[0.9],[1.2],[-0.9]])


time,vitXref,posXref,vitThref,posThref=TrajRef(Xdep,Xfin,tfin,t1,t2,STADE,R,timestep=dt)
t=1.5
X_pt0,X0,Th_pt0,Th0=CalcConsigne(t,time,vitXref,posXref,vitThref,posThref)
print("Vmax=",np.sqrt(np.sum(X_pt0*X_pt0)))

```

```python

"Résolution de l'équation différentielle"

#Conditions intiales
Y0=np.zeros(12)
Y0[0:3]=Xdep.transpose();


#résolution
Ysol=spint.odeint(EqDiff,Y0,time,args=(STADE,R,mA,Sinv,L,G,g,Kpos,Kvit,time,vitXref,posXref,vitThref,posThref,Force_pert))


#Extraction des valeurs à partir des solutions
x=Ysol[:,0];      y=Ysol[:,1];       z=Ysol[:,2];
xpt=Ysol[:,3];    ypt=Ysol[:,4];     zpt=Ysol[:,5];
TH1=Ysol[:,6];    TH2=Ysol[:,7];     TH3=Ysol[:,8];
TH1pt=Ysol[:,9]; TH2pt=Ysol[:,10]; TH3pt=Ysol[:,11];


#Calcul des forces de tension sur les 3 treuils
xptpt=np.gradient(xpt,dt)
yptpt=np.gradient(ypt,dt)
zptpt=np.gradient(zpt,dt)
F1=np.zeros(len(time))
F2=np.zeros(len(time))
F3=np.zeros(len(time))
for k in range(len(time)):
    X=np.array([[x[k]],[y[k]],[z[k]]])
    Xptpt=np.array([[xptpt[k]],[yptpt[k]],[zptpt[k]]])
    P=CalcPXTH(X,STADE,R)
    F=mA/R*np.dot(np.linalg.inv(np.transpose(P)),Xptpt+np.array([[0],[0],[g]])-Force_pert(time[k])/mA)
    F1[k]=F[0]; F2[k]=F[1]; F3[k]=F[2];


#Calcul de la tension aux bornes des moteurs
U1=np.zeros(len(time))
U2=np.zeros(len(time))
U3=np.zeros(len(time))
for k in range(len(time)):
    X_pt0,X0,Th_pt0,Th0=CalcConsigne(time[k],time,vitXref,posXref,vitThref,posThref)
    U1[k]=Kpos*(Th0[0]-TH1[k])+Kvit*(Th_pt0[0]-TH1pt[k])
    U2[k]=Kpos*(Th0[1]-TH2[k])+Kvit*(Th_pt0[1]-TH2pt[k])
    U3[k]=Kpos*(Th0[2]-TH3[k])+Kvit*(Th_pt0[2]-TH3pt[k])


```

```python
def plan(seuil):
    i=[]
    j=[]
    for x in arange(0,D,0.003):
        for y in arange(0,H,0.003):
            M=[x,y]
            if (sin(th1(M)+th2(M)))!=0:
                if (m*g*(cos(th2(M))/sin(th1(M)+th2(M))))>seuil or (m*g*(cos(th1(M))/sin(th1(M)+th2(M))))>seuil:
                    i.append(M[0])
                    j.append(M[1])
    k=100*(H-min(j))/H
    return [i,j,k]

X= plan(5)[0]
Y= plan(5)[1]

p2=plt.plot(X,Y)
plt.axis([0,D,0.5,H])
plt.show()

print(plan(5)[2])
```