Étude de positionnement

Matthieu ZAJAC

SCEI : 33870

Introduction

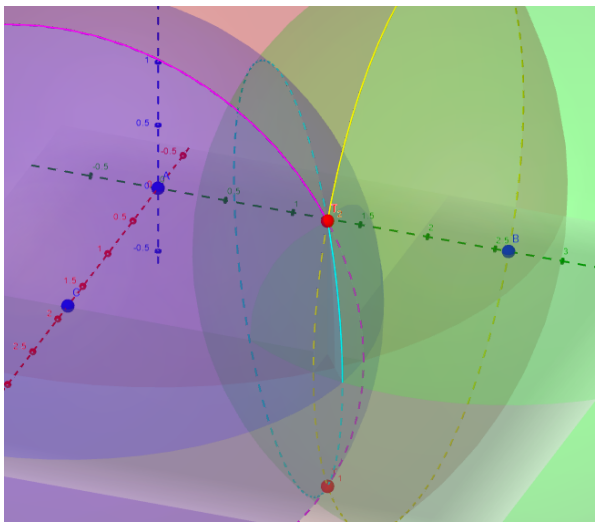
Problème

Objectif : Arbitrage réellement impartial

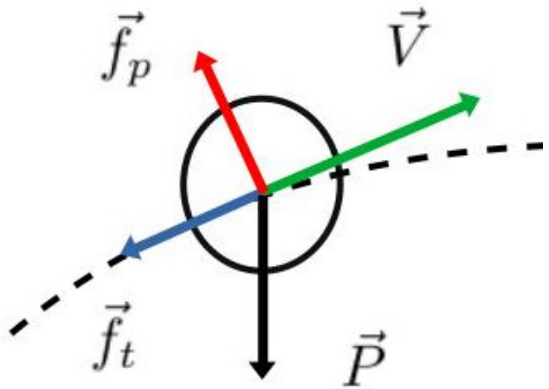
(Mais où est tombée la balle ?)

Solutions

Trilatération



Mécanique



Vidéo

Plan de l'étude

Trilatération UWB :

- Choix des composants
- Trilatération
- Mesure de distance
- Mise en place du système (connectique + programme Arduino)
- Récupération et traitement des données sur python
- Mise en situation

Analyse mécanique :

- Mise en équation du problème
- Détermination des constantes du problème
- Résolution des équations

Analyse vidéo :

- Pointage automatique
- Affichage de la trajectoire en cm
- Validation du modèle physique

Plan de l'étude

Trilatération UWB :

- Choix des composants
- Trilatération
- Mesure de distance
- Mise en place du système (connectique + programme Arduino)
- Récupération et traitement des données sur python
- Mise en situation

Analyse mécanique :

- Mise en équation du problème
- Détermination des constantes du problème
- Résolution des équations

Analyse vidéo :

- Pointage automatique
- Affichage de la trajectoire en cm
- Validation du modèle physique

Plan de l'étude

Trilatération UWB :

- Choix des composants
- Trilatération
- Mesure de distance
- Mise en place du système (connectique + programme Arduino)
- Récupération et traitement des données sur python
- Mise en situation

Analyse mécanique :

- Mise en équation du problème
- Détermination des constantes du problème
- Résolution des équations

Analyse vidéo :

- Pointage automatique
- Affichage de la trajectoire en cm
- Validation du modèle physique

Plan de l'étude

Trilatération UWB :

- Choix des composants
- Trilatération
- Mesure de distance
- Mise en place du système (connectique + programme Arduino)
- Récupération et traitement des données sur python
- Mise en situation

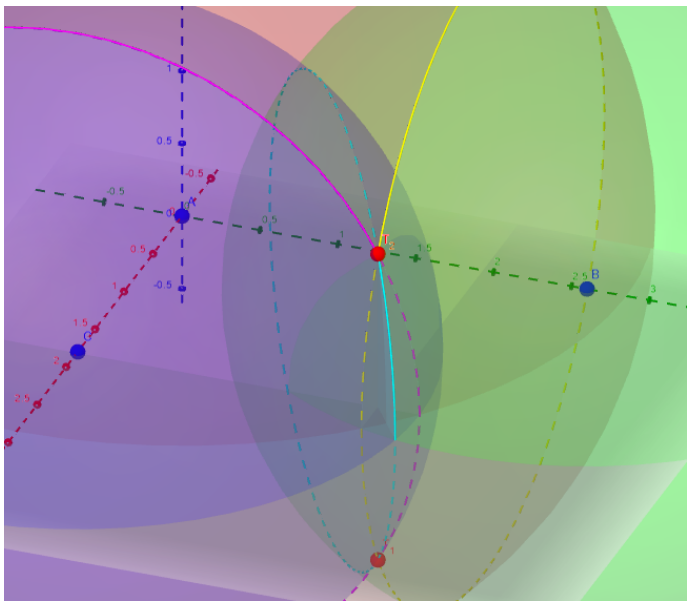
Analyse mécanique :

- Mise en équation du problème
- Détermination des constantes du problème
- Résolution des équations

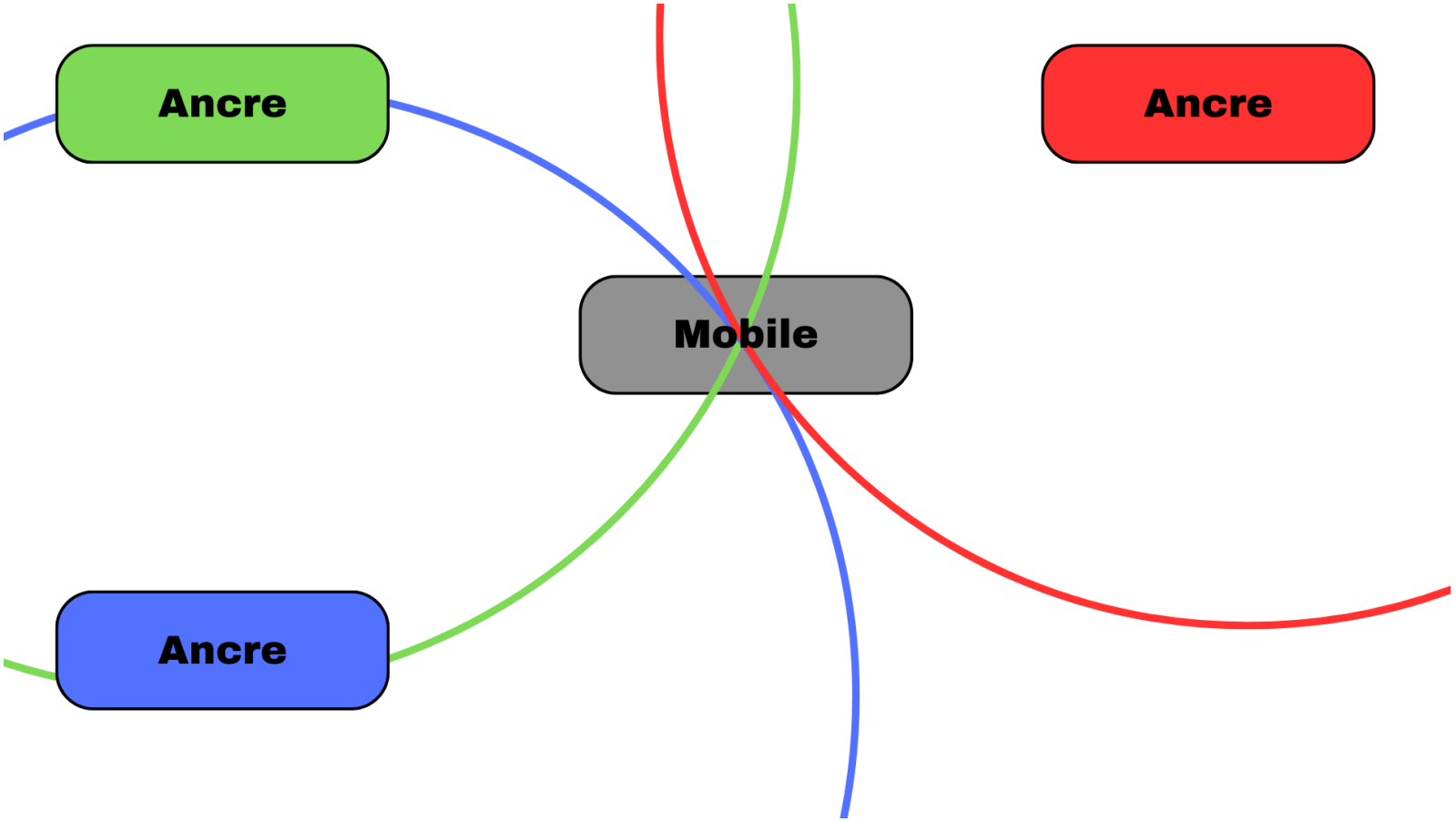
Analyse vidéo :

- Pointage automatique
- Affichage de la trajectoire en cm
- Validation du modèle physique

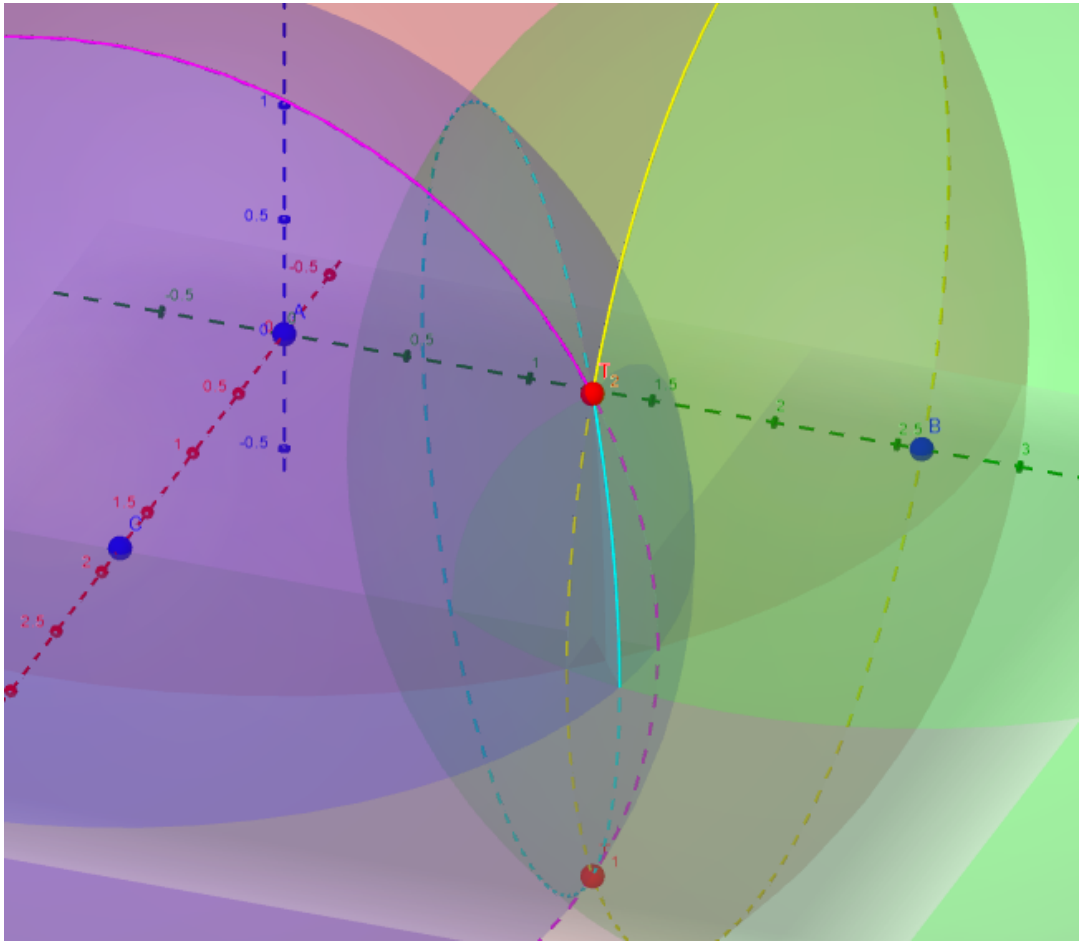
Trilatération (UWB)



Trilatération



Trilatération 3D



Trilatération 3D

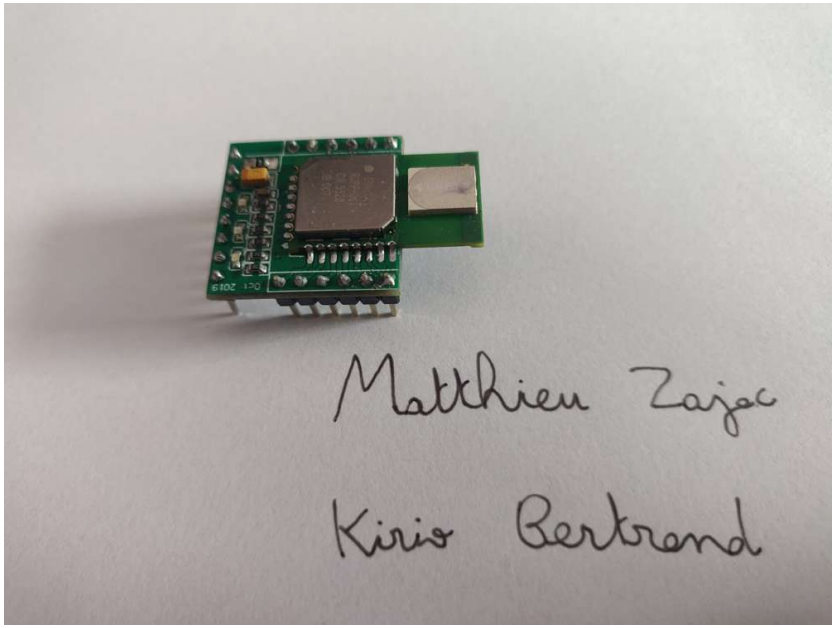
intersection de 3 sphères :

$$\begin{cases} x^2 + y^2 + z^2 = R_1^2 \\ x^2 + (y - y_2)^2 + z^2 = R_2^2 \\ (x - x_3)^2 + y^2 + z^2 = R_3^2 \end{cases}$$

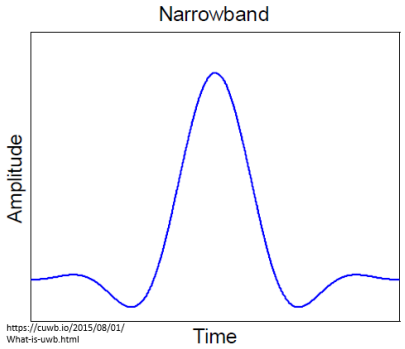
$$\begin{aligned} A_1 &: (0, 0, 0) \\ A_2 &: (0, y_2, 0) \\ A_3 &: (x_3, 0, 0) \end{aligned}$$

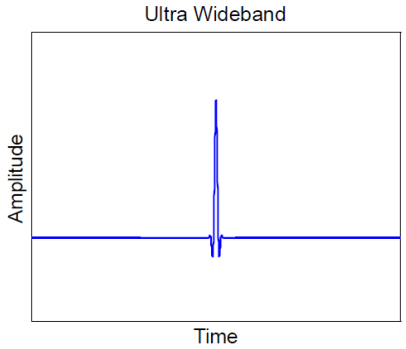
$$\begin{cases} z = \pm\sqrt{-\left(\frac{R_1^2 - R_3^2}{2x_3} + \frac{x_3}{2}\right)^2 - \left(\frac{R_1^2 - R_2^2}{2y_2} + \frac{y_2}{2}\right)^2 + R_1^2} \\ y = \frac{R_1^2 - R_2^2}{2y_2} + \frac{y_2}{2} \\ x = \frac{R_1^2 - R_3^2}{2x_3} + \frac{x_3}{2} \end{cases}$$

UWB

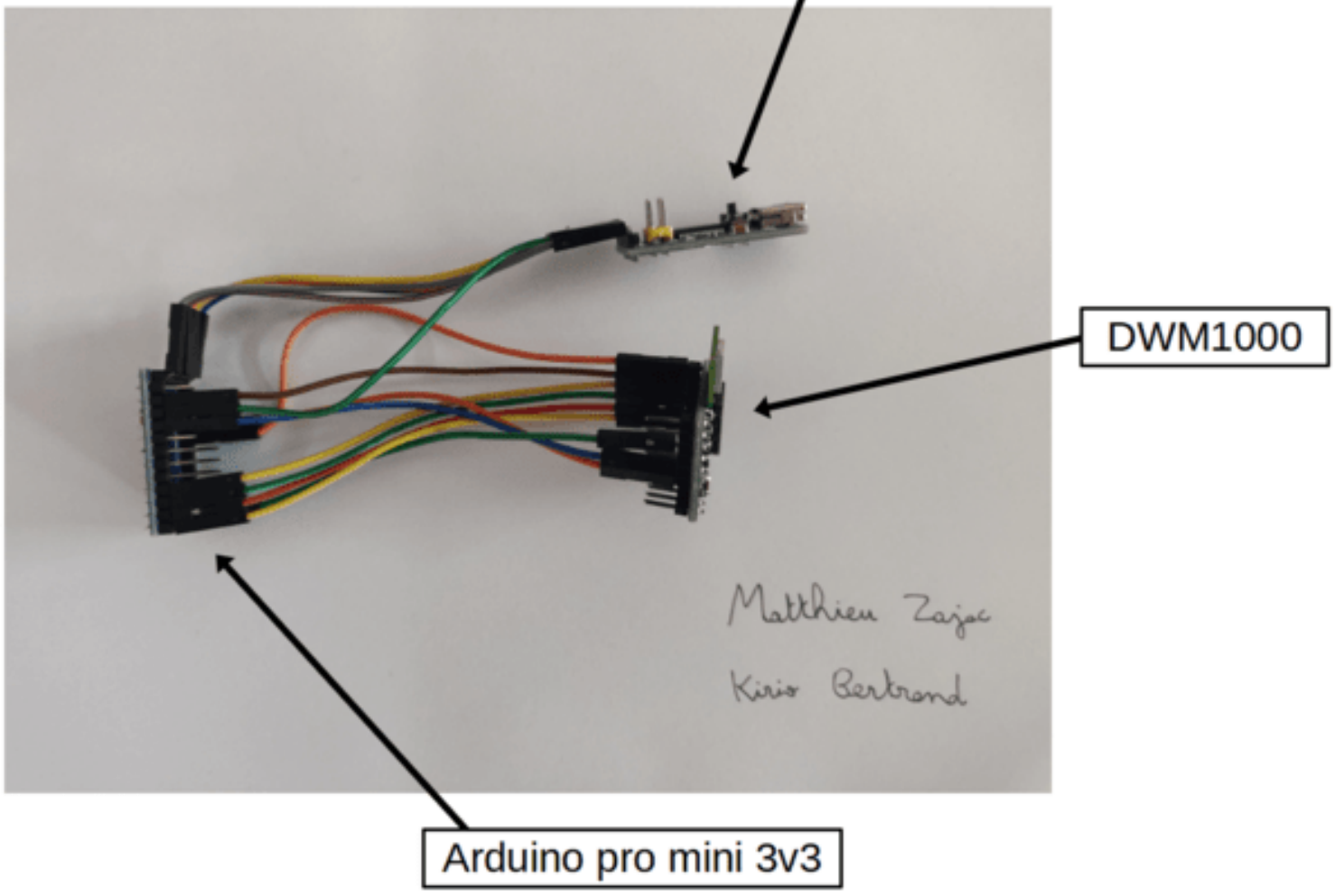


- Fréquences : largeur de 500 MHz entre 3.1 et 10.6 GHz
- Durée d'impulsion : 2 ns
- Amplitude proche du bruit de fond
- Précision entre 3 à 28 cm

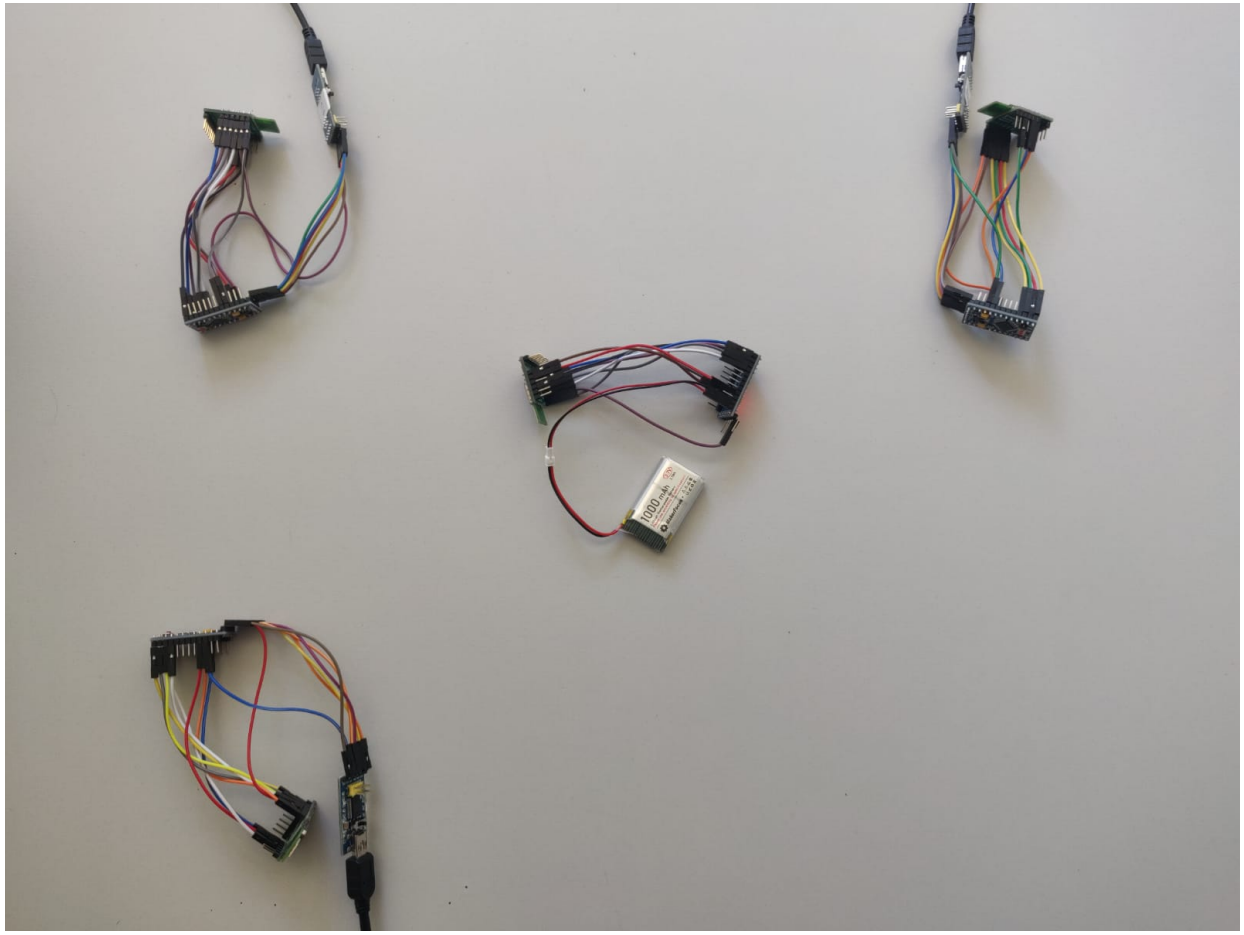




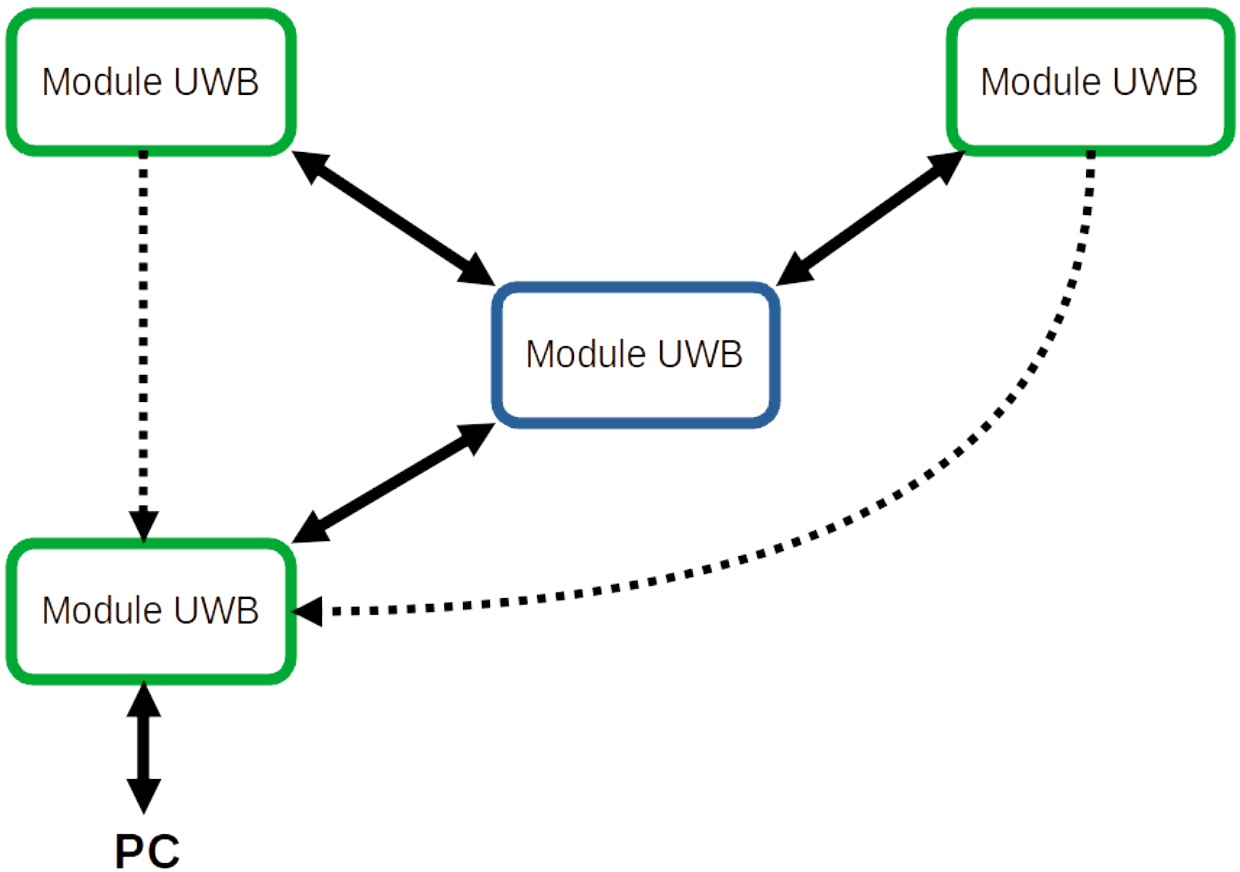
Module


Convertisseur USB - Serial
DWM1000
Arduino pro mini 3v3
Matthieu Zajac
Kirio Bertrand

Système

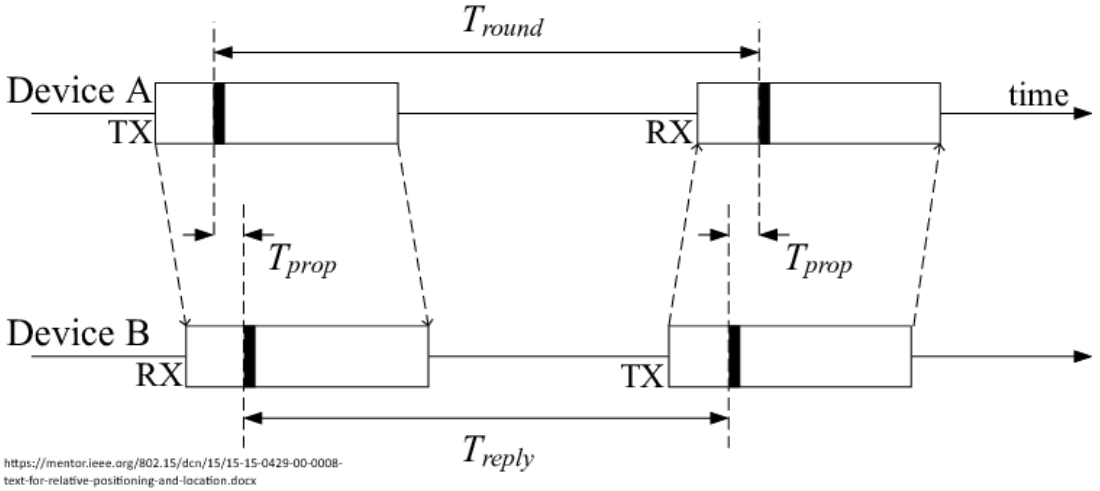


Communications



Algorithme

Single-sided two-way ranging :



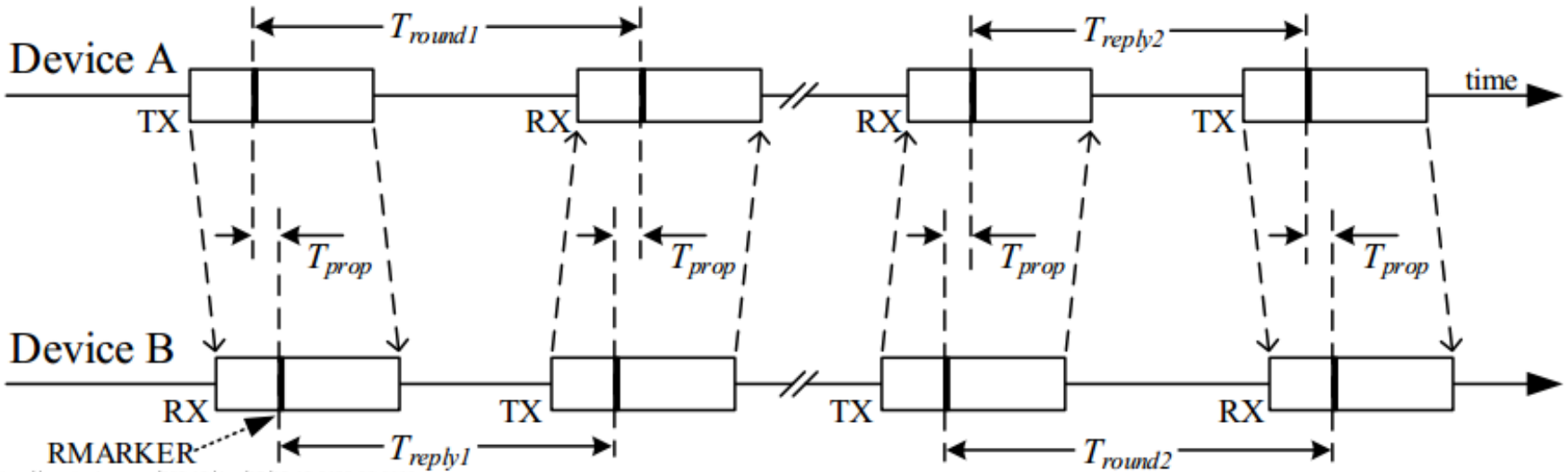
TWR (Two way ranging) :

- 2 capteurs
- Position relative (distance)
- (Apple Airtag)

$$\widehat{T_{prop}} = \frac{1}{2}(T_{round} - T_{reply})$$

Algorithme

Asymmetric double sided two-way ranging

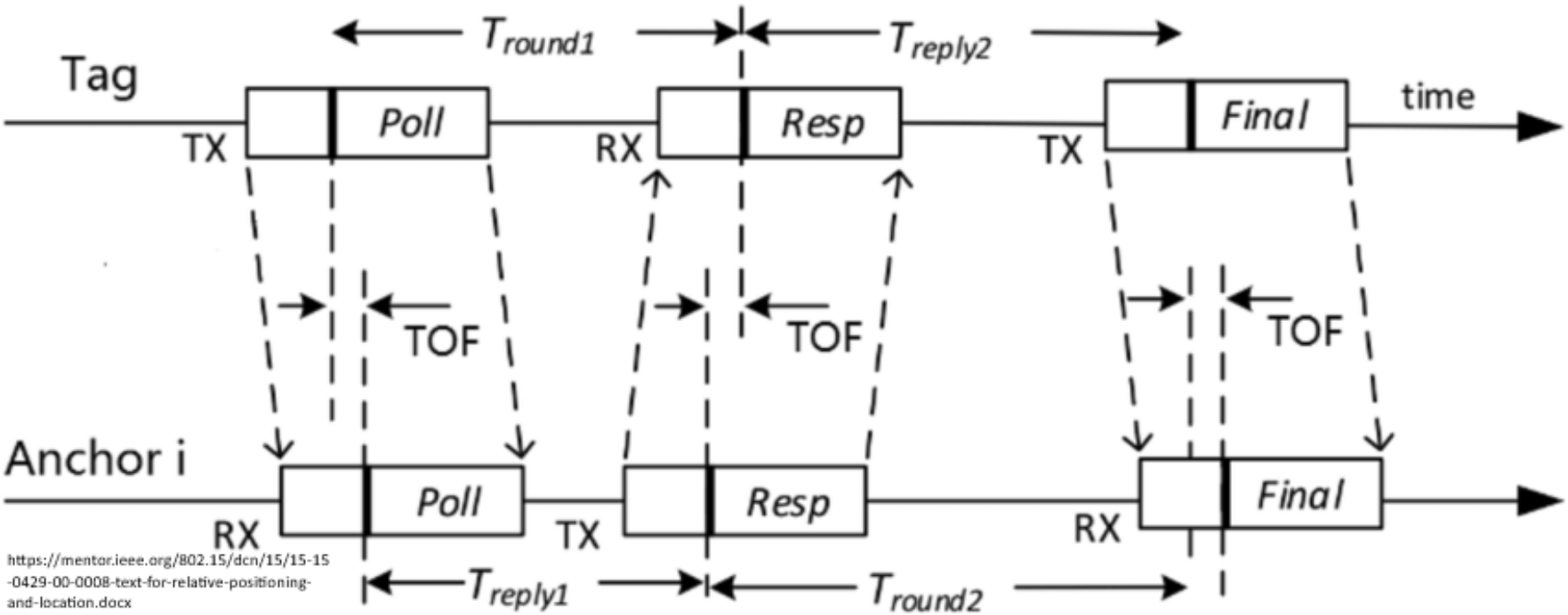


https://mentor.ieee.org/802.15/dcn/15/15-15-0429-00-0008-text-for-relative-positioning-and-location.docx

$$\widehat{T_{prop}} = \frac{T_{round1} * T_{round2} - T_{reply1} * T_{reply2}}{T_{round1} + T_{round2} + T_{reply1} + T_{reply2}}$$

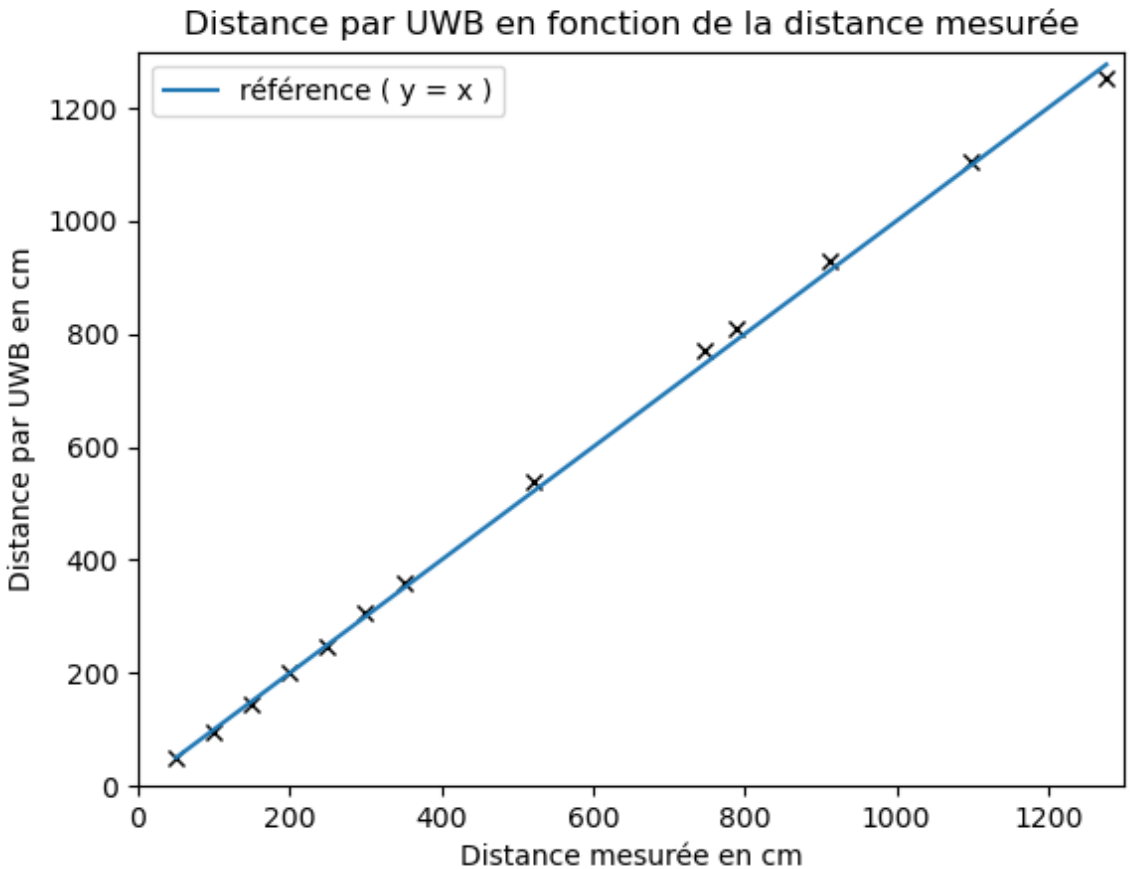
Algorithme

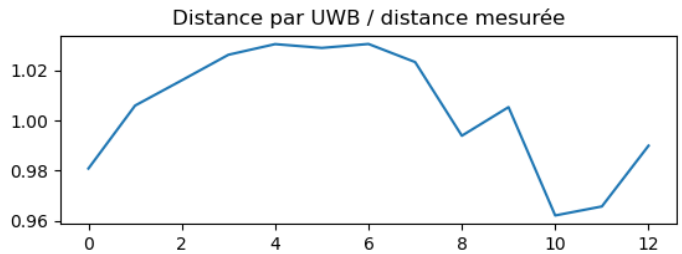
On combine les 2 signaux centraux



Mesures de distances

Mesures de distances

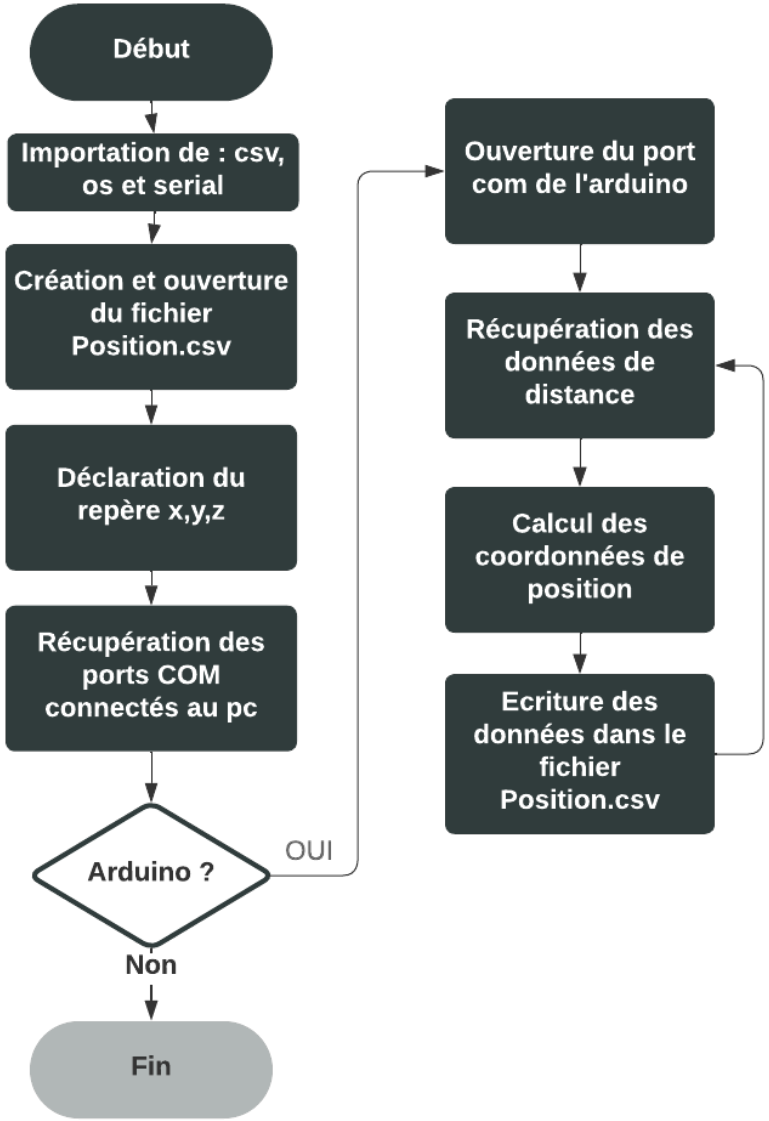


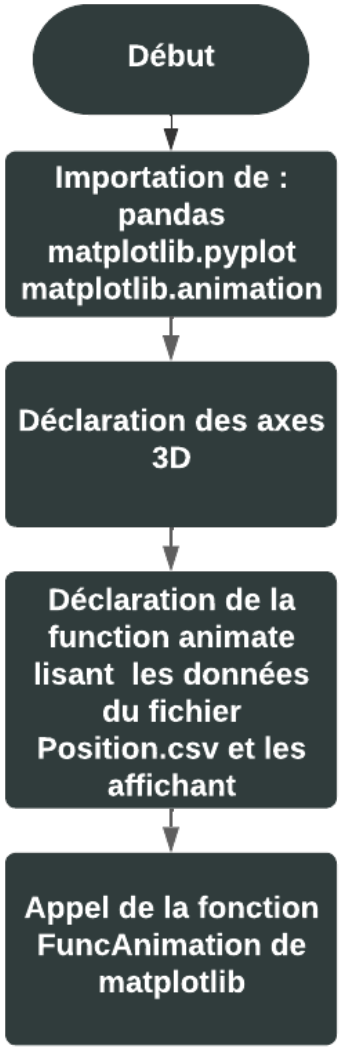


Les valeurs mesurées sont cohérentes

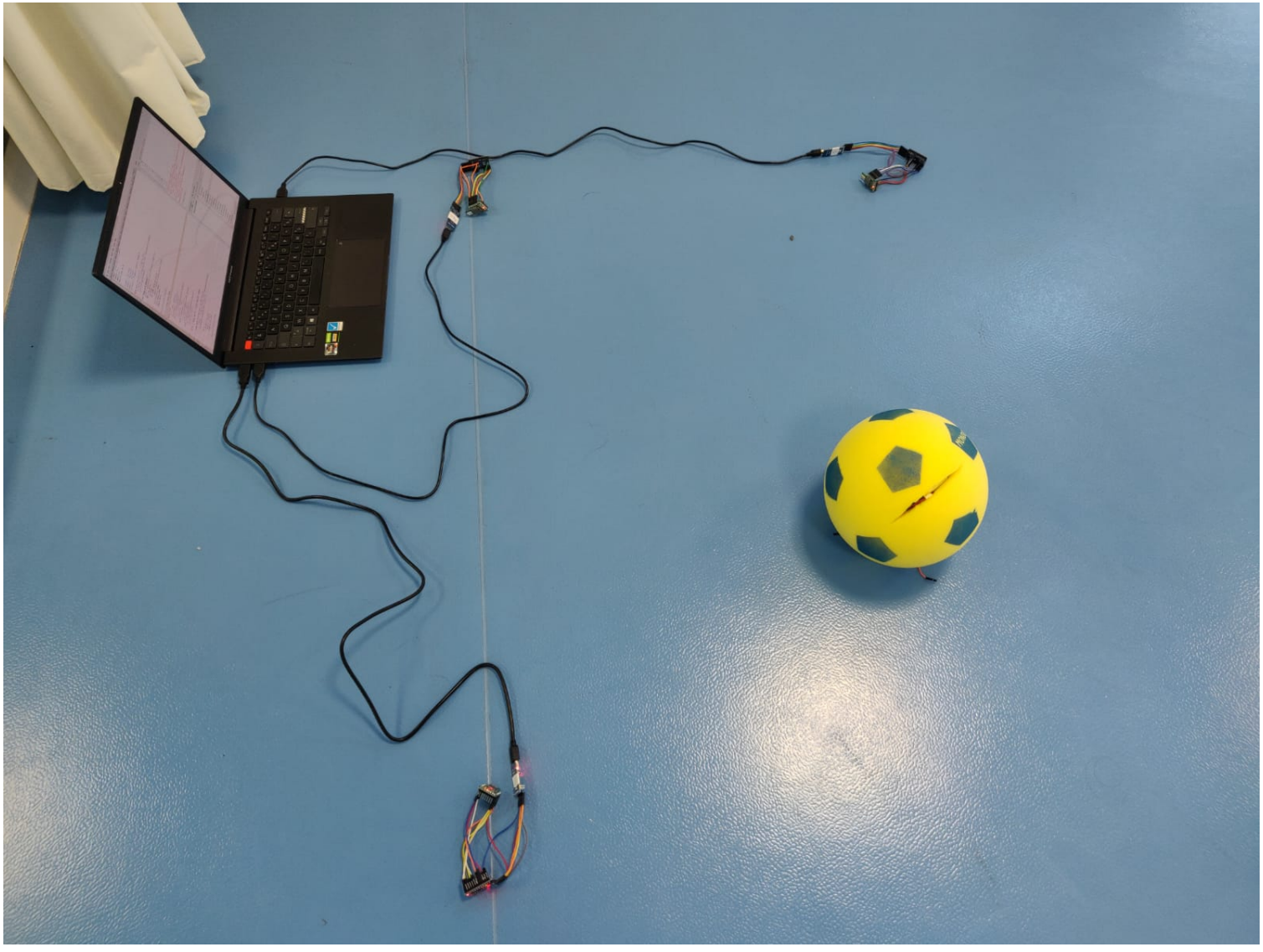
Précision suffisante

Calcul et affichage des résultats

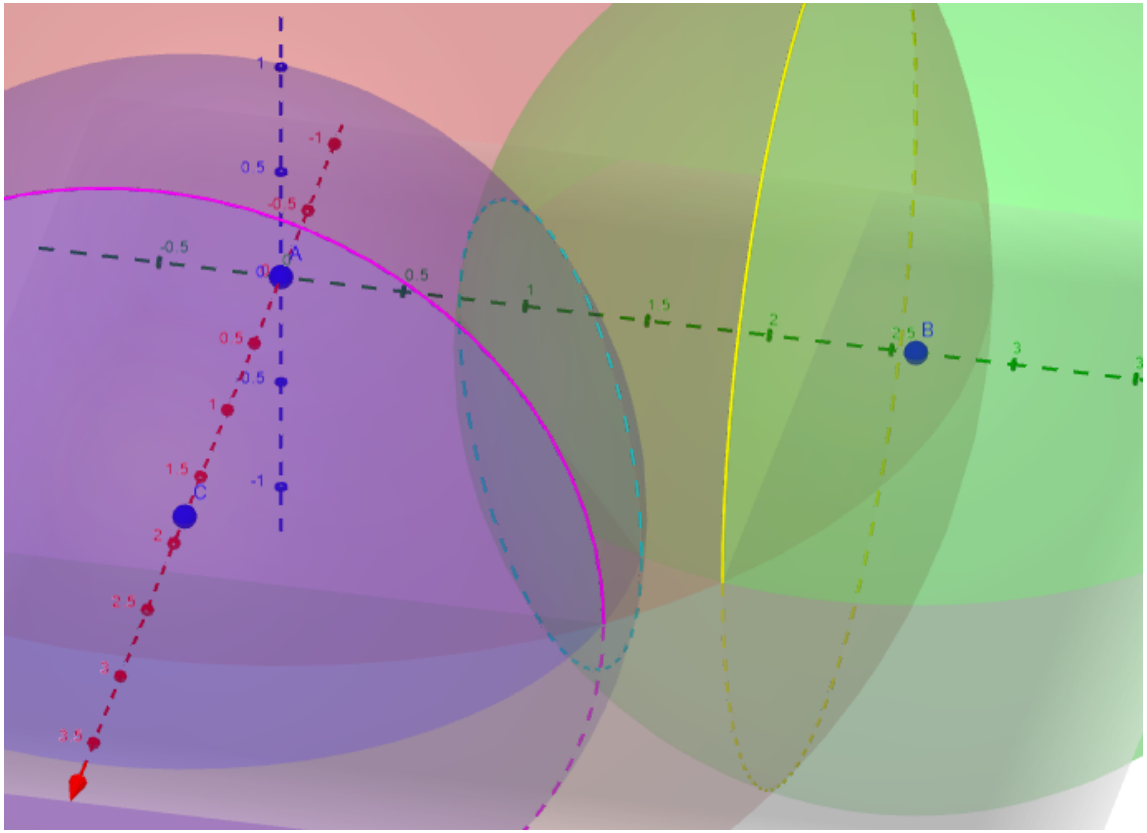




Mesures de positions



Problème



y2=2.6m

x3=1.8m

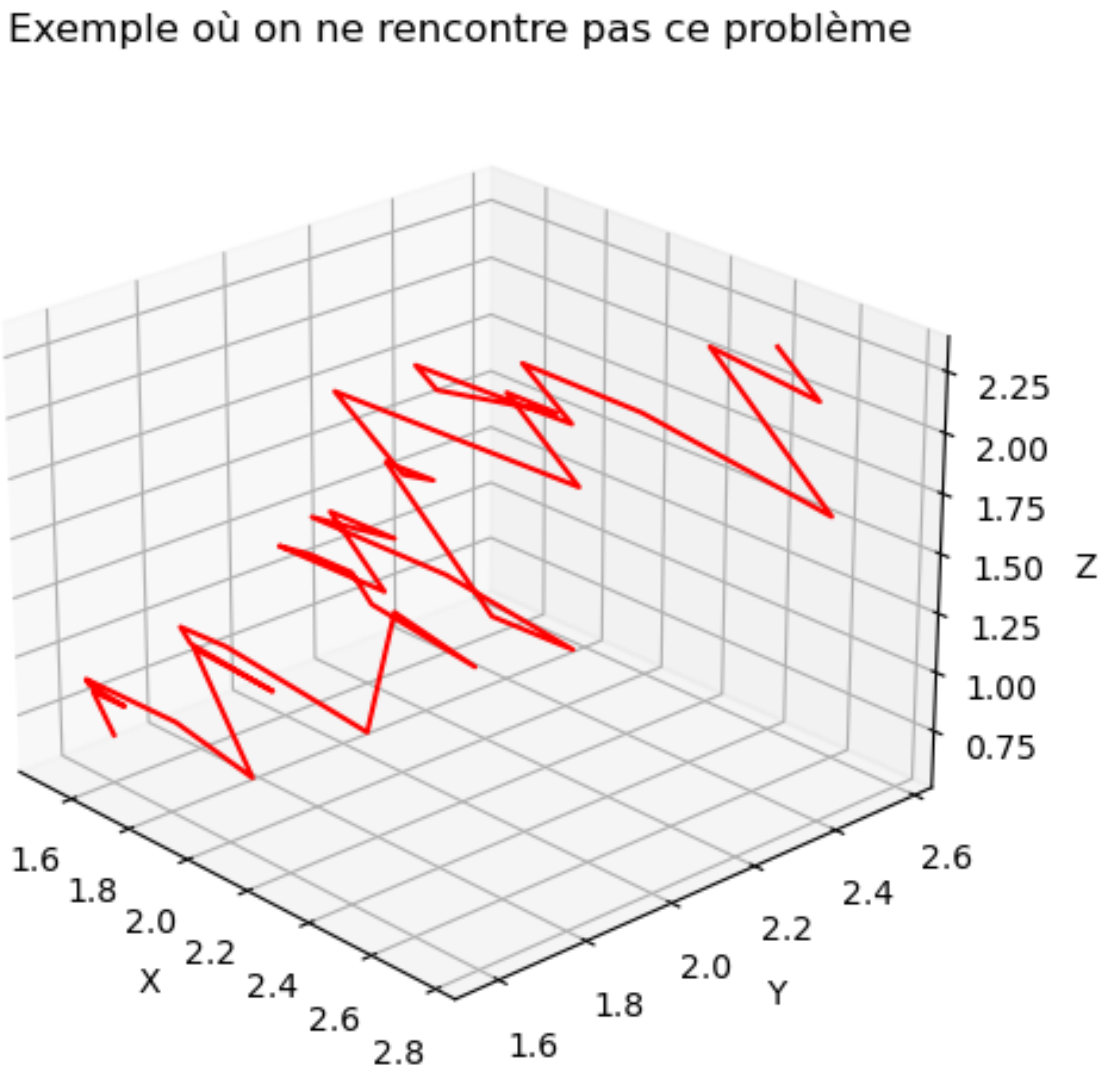
D1=2.80m

D2=1.85m

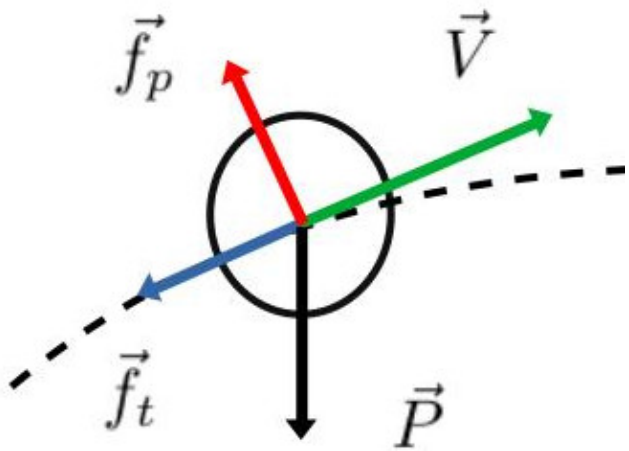
D3=1.85m

Problème : aucune intersection

Lorsque cela fonctionne



Mécanique



Mécanique

système : balle de masse m

Repère : (O,x,y,z)

référentiel : terrestre supposé galiléen

Forces :

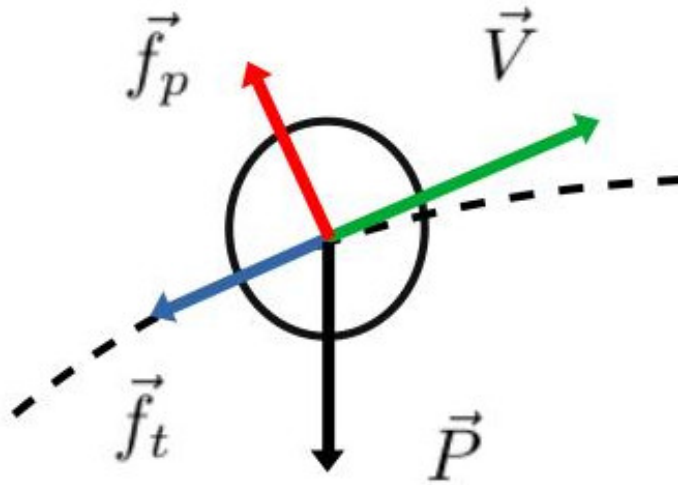
$\vec{P} = m\vec{g}$

force de trainée : $\vec{f_T} = -\frac{1}{2}\rho S C_x \|\vec{V}\|\vec{V}$

force de portance : $\vec{f_P} = \mu\vec{\omega} \wedge \vec{V}$

Principe fondamental de la dynamique :

$m\vec{a} = \vec{P} + \vec{f_T} + \vec{f_P}$



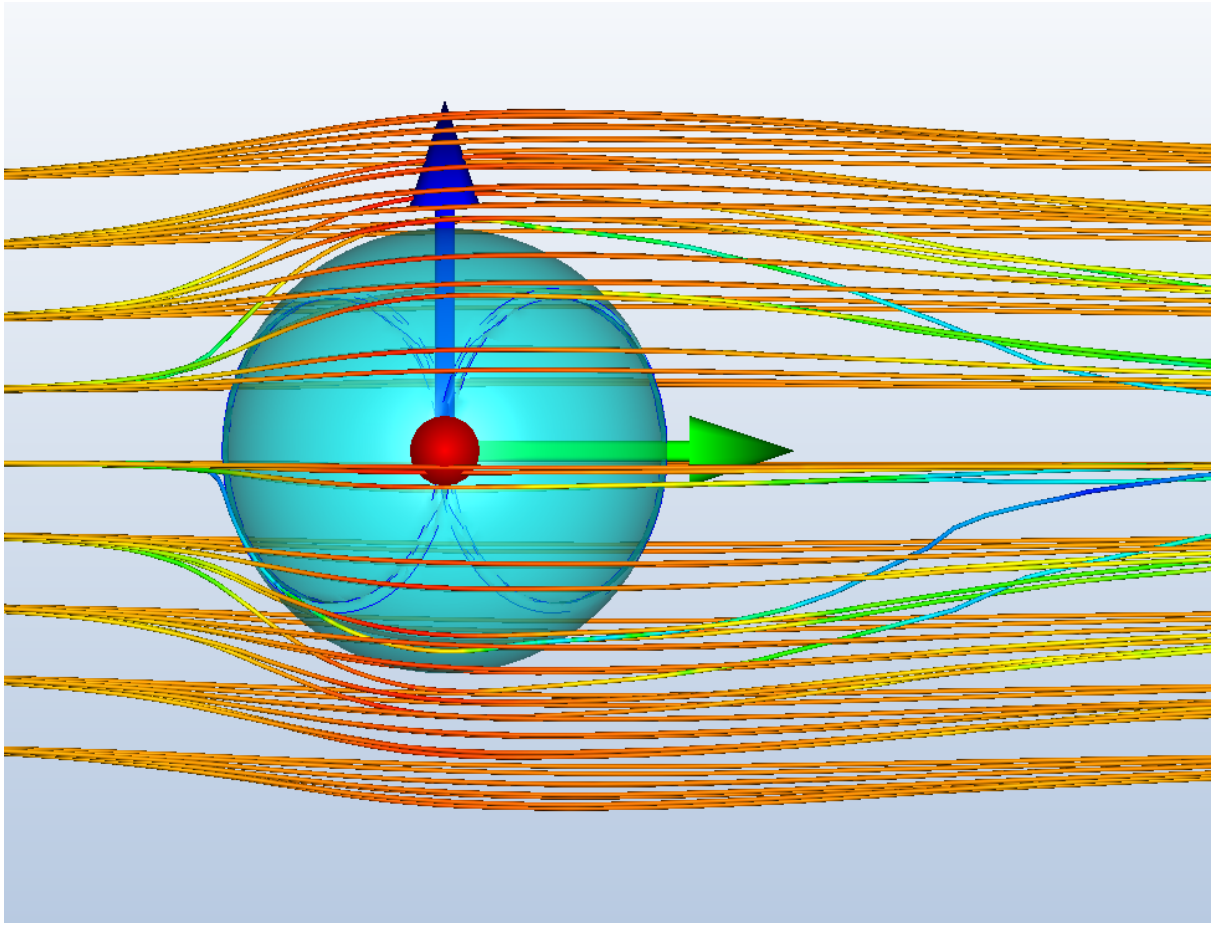
Coefficients de trainée et de portance

Nombre de Reynolds : $Re = \frac{\rho V D}{\eta}$

Coefficient de portance : μ en $kg.rad^{-1}$ caractérisé par la balle.

Coefficient de traînée : Cx, sans dimension.

Détermination de Cx



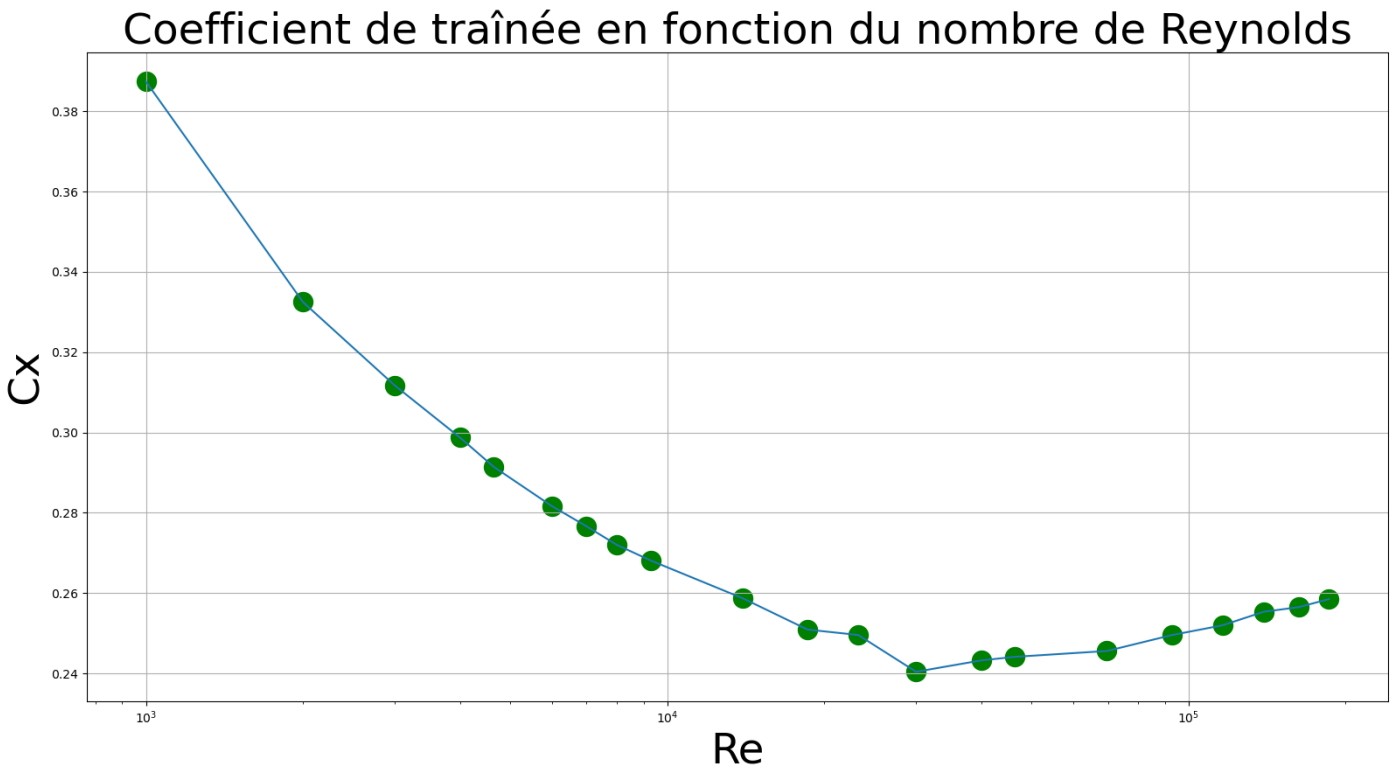
Caractéristiques :

Diamètre : 66 mm

Rugosité : 1 mm

Vitesse : 100 km/h

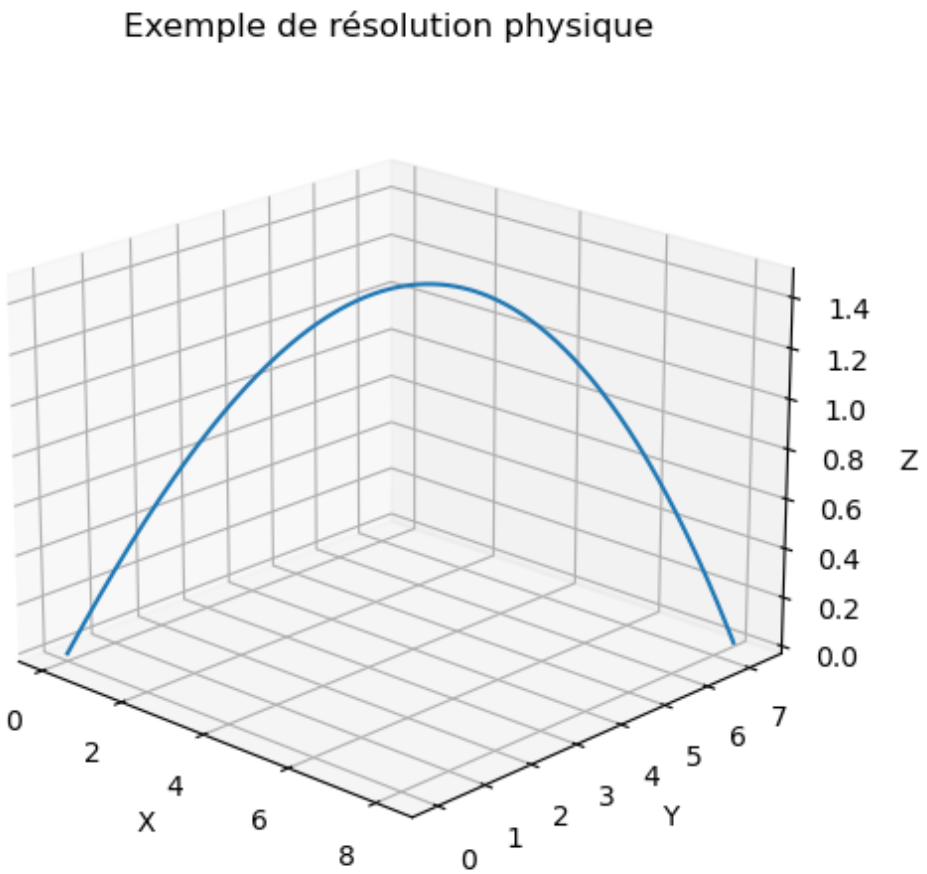
Caractérisation de Cx



Résolution de l'équation du mouvement

- Projection de l'équation dans la base de coordonnées cartésiennes.
- Résolution du système différentiel non linéaire sur la vitesse sous Python en utilisant le schéma d'euler explicite.
- Intégration numérique de la vitesse.
- Trajectoire avec matplotlib.

Résolution sous python



$Cx = 0.17$

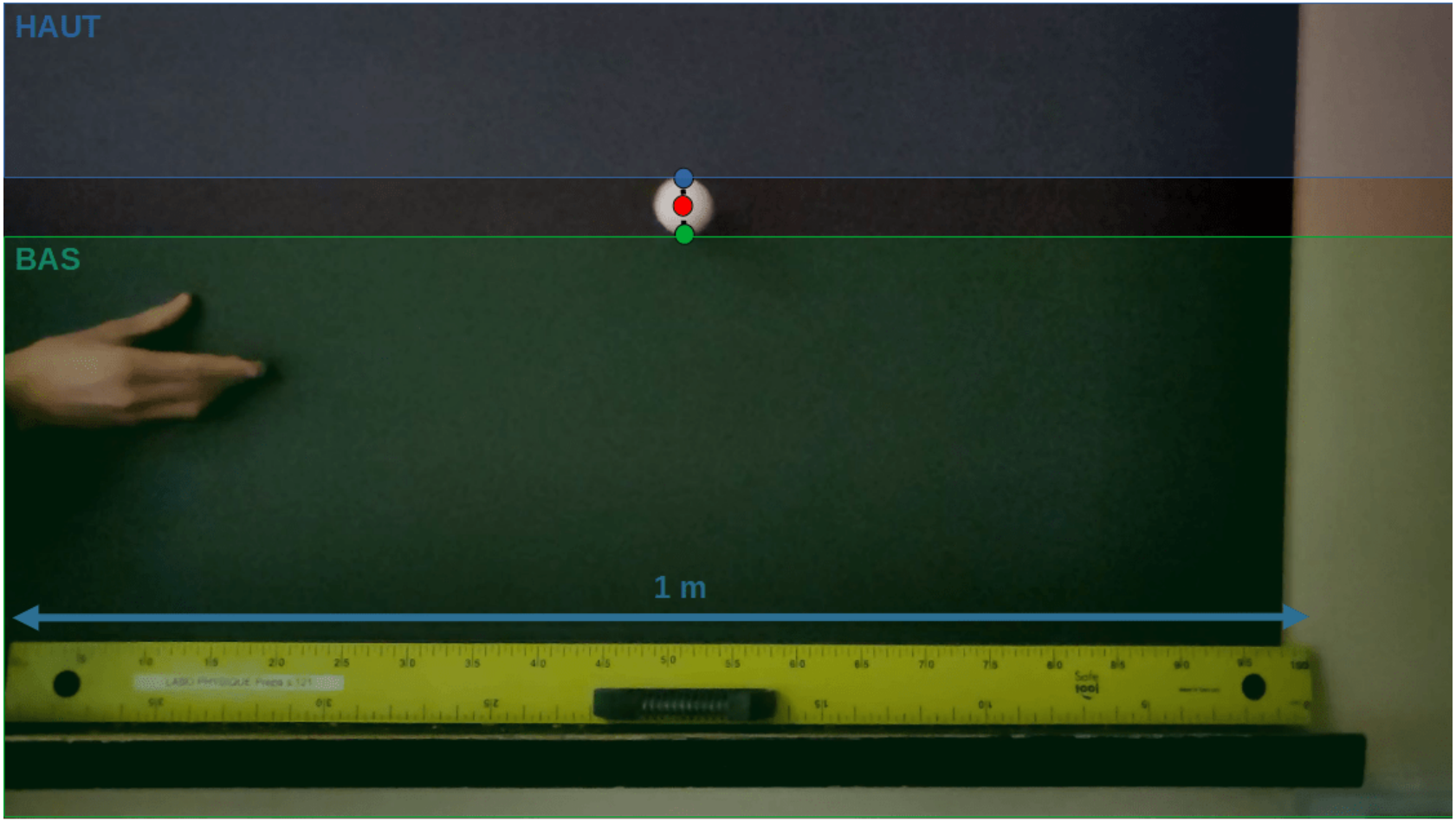
$Vx0 = 9$ m/s

$Vy0 = 7$ m/s

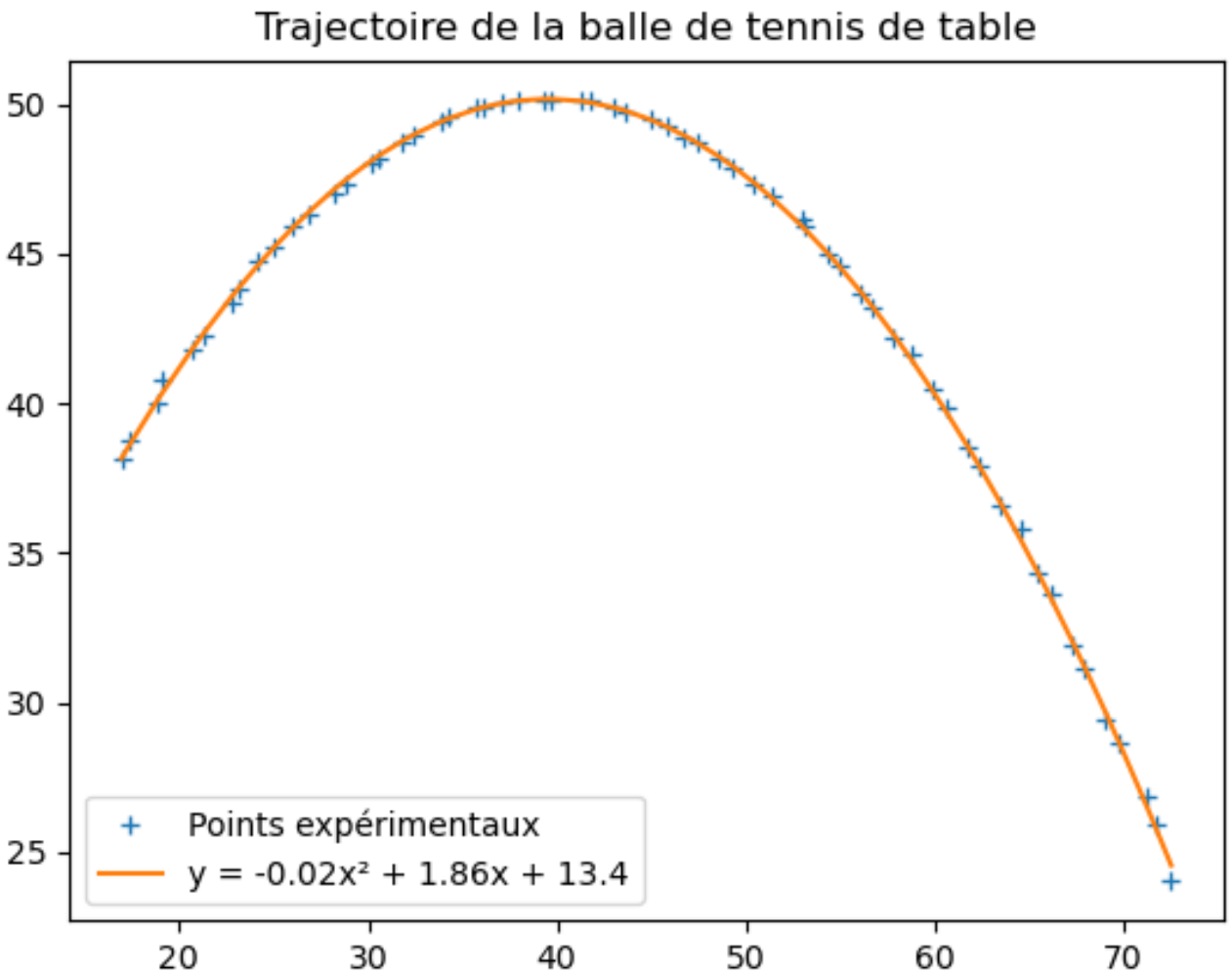
$vz0 = 5$ m/s

Analyse vidéo

Principe

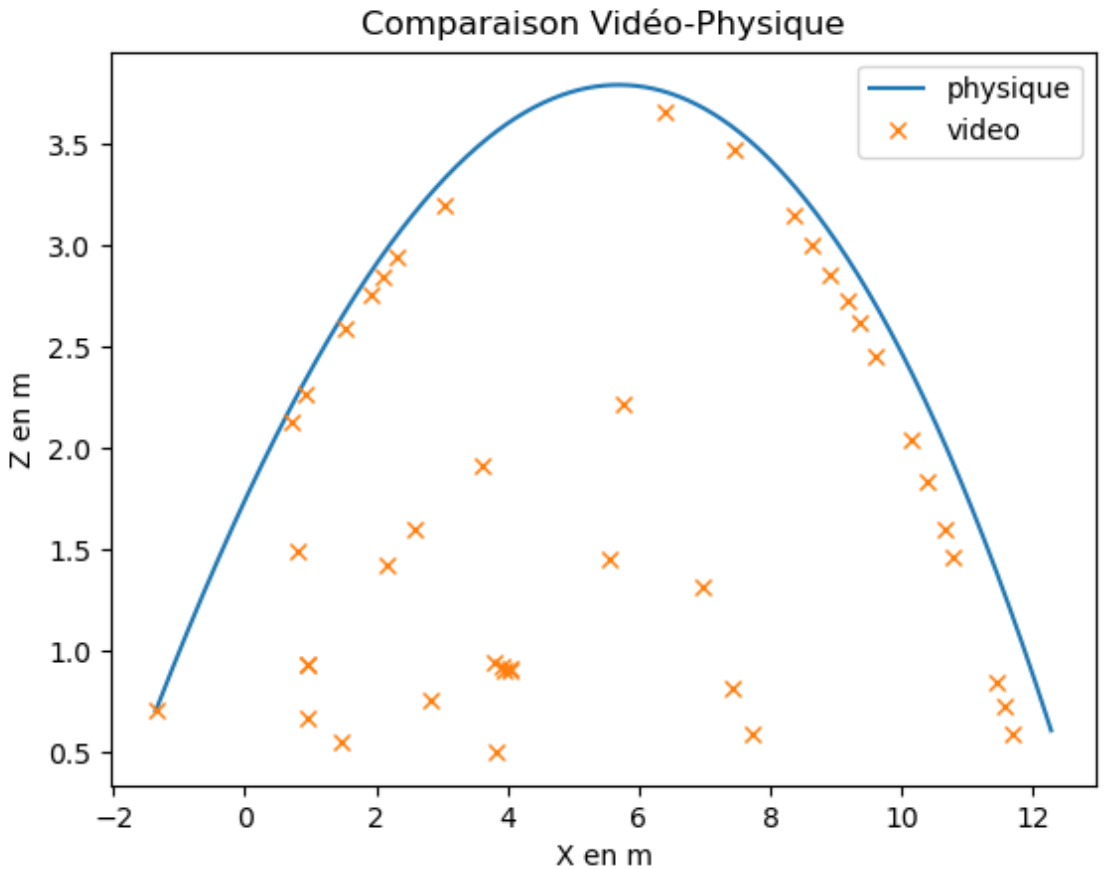


Résultats


Trajectoire de la balle de tennis de table
50
45
40
35
30
25
20
30
40
50
60
70
Points expérimentaux
y = -0.02x² + 1.86x + 13.4

Validation du modèle physique

Validation du modèle physique



$$Vx0 = 9.1 \text{ m/s}$$
$$Vy0 = 0 \text{ m/s}$$
$$Vz0 = 7.6 \text{ m/s}$$

Bilan

Sections	Objectifs	Résultats
UWB	Mesure de position en temps réel	Mesures de distances cohérentes et assez précises, algorithme de trilatération à revoir. Augmenter le nombre d'ancres ?
Mécanique	Modélisation de la trajectoire d'une balle Détermination des constantes du problème	Caractérisation de Cx en fonction de Re Modèle cohérent
Vidéo	Pointage automatique	Longueur focale et inclinaison non prises en compte Très sensible à l'arrière-plan

Annexes

Two-way ranging

$T_{round1} = 2T_{prop} + T_{reply1}$

$T_{prop} = \frac{1}{2}(T_{round1} - T_{reply1})$

En pratiques les horloges internes de A et B sont plus ou moins rapides

On suppose que ces horloges tournent respectivement à ka et kb fois la fréquence d'une horloge idéale

On a donc :

$Tp_{round1} = k_a T_{round1}$

$Tp_{reply2} = k_a T_{reply2}$

$Tp_{round2} = k_b T_{round2}$

$Tp_{reply1} = k_b T_{reply1}$

Calcul de l'erreur :

$Er = \frac{1}{2}((k_a - 1)T_{round1} - (k_b - 1)T_{reply1})$

Two-way ranging

$$T_{reply2} = \frac{Tp_{reply2}}{k_a} \text{ et } T_{reply2} = \frac{Tp_{reply2}}{k_b}$$

$$T_{round1} = 2T_{prop} + \frac{Tp_{reply1}}{k_b}$$

$$Tp_{round1} = 2k_a T_{prop} + \frac{k_a Tp_{reply1}}{k_b}$$

$$Tp_{round2} = 2k_a T_{prop} + \frac{k_b Tp_{reply2}}{k_a}$$

En multipliant Tp_{round1} et Tp_{round2} on obtient :

$$Tp_{round1} Tp_{round2} =$$

$$4k_a k_b T_{prop}^2 + \frac{k_a Tp_{reply1}}{k_b} \frac{k_b Tp_{reply2}}{k_a} + 2k_b T_{prop} \frac{k_a Tp_{reply1}}{k_b} + 2k_a T_{prop} \frac{k_b Tp_{reply2}}{k_a}$$

$$Tp_{round1} Tp_{round2} =$$

$$4k_a k_b T_{prop}^2 + Tp_{reply2} Tp_{reply1} + 2T_{prop}(k_a Tp_{reply1} + k_b Tp_{reply2})$$

$$\frac{Tp_{round1} Tp_{round2} - Tp_{reply1} Tp_{reply2}}{Tp_{round1} + Tp_{reply2}} = 2T_{prop} k_b \approx 2T_{prop}$$

$$\frac{Tp_{round1} Tp_{round2} - Tp_{reply1} Tp_{reply2}}{Tp_{round2} + Tp_{reply1}} = 2T_{prop} k_a \approx 2T_{prop}$$

2 estimations différentes de T_{prop}. En les reliant on obtient :

$$ToF = \frac{Tp_{round1} Tp_{round2} - Tp_{reply1} Tp_{reply2}}{Tp_{round1} + Tp_{round2} + Tp_{reply1} + Tp_{reply2}}$$

Projection

Regression par projection

Orthonormaliser la base $(\tilde{1}, \tilde{x}, \tilde{x^2})$ de l'ensemble des fonctions polynomiales de degré 2 grâce à Gram-Schmidt :

$u'_1 = \tilde{1} \quad u'_2 = \tilde{x} - \frac{<\tilde{x},\tilde{1}>}{<\tilde{1},\tilde{1}>}\tilde{1} \quad u'_3 = \tilde{x^2} - \frac{<\tilde{x^2},\tilde{1}>}{<\tilde{1},\tilde{1}>}\tilde{1} - \frac{<\tilde{x^2},u'_2>}{<u'_2,u'_2>}u'_2$

On normalise la base obtenue : $(\frac{u'_1}{||u'_1||}, \frac{u'_2}{||u'_2||}, \frac{u'_3}{||u'_3||})$

Programme python capable de réaliser cet algorithme pour les polynômes de degré n.

On prend des points d'un polynôme défini en les décalant.

Interpolation de lagrange en ces points.

Projection du polynôme de degré N+1 obtenu dans la base trouvée précédemment.

Interpolation et Régression

```python
from math import *
import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate as sc

def P(l,x:float):   # l est la liste des coefficients du plus bas au plus haut degré
    res=0
    for n,i in enumerate(l):
        res+=i*x**n    # evalue l'équation polynomiale en x
    return res

def orthonormal(n):   # renvoie la base orthonormée des polynomes de degré n-1
    base=[]
    for i in range(n+1):
        base.append(np.array([0]*i + [1] + [0]*(n-i)))      # on écrit la base canoni

    def ps(l1,l2):  # définie le produit scalaire utilisé
        res=0
        for i in range(n+1):
            res+=P(l1,i)*P(l2,i)
        return res

    ortho=[base[0]]
    for i in range(1,n+1):  # applique Gram-Schmidt pour trouver une base orthogonale
        add=base[i]
        k=i-1
        while k!=-1:
            add=np.add(add,-(ps(base[i],ortho[k])/ps(ortho[k],ortho[k]))*ortho[k])
            k-=1
        ortho.append(add)

    normal=[]
    for i in range(n+1):
        normal.append(ortho[i]/sqrt(ps(ortho[i],ortho[i])))     # normalise les vecte

    return normal

def points(l,N,h):   # renvoie une liste de points de la fonction polynomiale considé
    xtab=np.linspace(-10,10,N)
    ytab=np.zeros(N)
    for k,x in enumerate(xtab):
        ytab[k]=P(l,x)
    for i,y in enumerate(ytab):
        ytab[i]= y + np.random.uniform(-h,h)
    return xtab,ytab
```

```python
def Lagrange(xtab,ytab):    # applique l'interpolation de Lagrange
    X=np.poly1d([1,0])
    P=0
    for i in range(len(ytab)):
        Li=1
        for j in range(len(ytab)):
            if i==j:
                continue
            else :
                Li=Li*((X-xtab[j])/(xtab[i]-xtab[j]))
        P+=Li*ytab[i]
    return P

N=20
xtab,ytab=points([3,0,-2],N,0.2)

L=Lagrange(xtab,ytab)   # L est du degré le plus au moins élevé
xtab2=np.linspace(-10,10,1000)
ylagrange=np.polyval(L,xtab2)

Llist=list(L)
Llist.reverse()     # on inverse l'ordre des coefficients

def ps1(l1,l2):     # produit scalaire pour la projection
    res=0
    for i in range(N):
        res+=P(l1,i)*P(l2,i)
    return res

def projection(n,L):        # projection dans la base précedemment trouvée
    base=orthonormal(n)
    res=np.zeros(3)
    for i in range(3):
        res+=ps1(base[i],L)*base[i]
    return res

proj=projection(2,Llist)
print(proj)

def points1(l,N):       # renvoie les points du polynome l
    xtab=np.linspace(-10,10,N)
    ytab=np.zeros(N)
    for k,x in enumerate(xtab):
        ytab[k]=P(l,x)
    return xtab,ytab

xtab1,ytab1=points1(proj,1000)

plt.figure()
plt.plot(xtab,ytab,'o')
#plt.plot(xtab1,ytab1)
plt.plot(xtab2,ylagrange)
#plt.plot(xtab2,P(Llist,xtab2))
plt.show()
```

importation des données et affichage

```
header1 = "x_value"
header2 = "y_value"
header3 = "z_value"
fieldnames = [header1, header2, header3]

with open('Position.csv', 'w') as csv_file:
    csv_writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
    csv_writer.writeheader()

def recup_port_Arduino() :
    ports = list(serial.tools.list_ports.comports())
    for p in ports:
        print(p)
        if 'COM8' in p.description :
            mData = serial.Serial(p.device,115200)
    print(mData.is_open)
    print(mData.name)
    return mData

def position(D1,D2,D3,y2,x3):
    y=(D1**2-D2**2+y2**2)/(2*y2)
    x=(D1**2-D3**2+x3**2)/(2*x3)
    z=np.sqrt(D1**2 - x**2 - y**2)
    return x,y,z

Data =recup_port_Arduino()

D1,D2,D3=False,False,False

while True:
    line1 = Data.readline()
    listeDonnees = line1.split()
    if listeDonnees[2]==b'2':
        D2 = float(listeDonnees[4].decode())
        print(f'D2 = {D2}')
    elif listeDonnees[2]==b'3':
        D3 = float(listeDonnees[4].decode())
        print(f'D3 = {D3}')
    elif listeDonnees[2]==b'm':
        D1 = float(listeDonnees[1].decode())
        print(f'D1 = {D1}')

    if D1 != False and D2 != False and D3!= False:

        with open('Position.csv', 'a') as csv_file:
            csv_writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
            x_value,y_value,z_value = position(D1,D2,D3,0.55,0.62)

            info = {
                header1: x_value,
                header2: y_value,
                header3: z_value
            }
            csv_writer.writerow(info)
```

```
import random
from itertools import count
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#plt.style.use('fivethirtyeight')

ax = plt.axes(projection='3d')

header1 = "x_value"
header2 = "y_value"
header3 = "z_value"

index = count()

def animate(i):
    data = pd.read_csv('Position.csv')
    x = data[header1]
    y = data[header2]
    z = data[header3]

    plt.cla()

    ax.scatter3D(x, y, z,'red')


    #plt.legend(loc='upper left')
    #plt.tight_layout()


ani = FuncAnimation(plt.gcf(), animate, interval=500)

plt.tight_layout()
plt.show()
```

Possible solution au problème

```python
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from scipy.optimize import minimize
import cmath

y2=2.6
x3=1.8


def position(D1,D2,D3):
    # Fonction à minimiser
    def objective(rayons):
        return rayons[0]**2+rayons[1]**2+rayons[2]**2

    #Fonction de contrainte
    def c(rayons):
        y=((D1+rayons[0])**2-(D2+rayons[1])**2+y2**2)/(2*y2)
        x=((D1+rayons[0])**2-(D3+rayons[2])**2+x3**2)/(2*x3)
        z2=(D1+rayons[0])**2 - x**2 - y**2
        return z2

    # Valeures initiales
    initial_guess = [0,0,0]

    # Minimisation
    cons = {'type':'ineq', 'fun': c}
    result = minimize(objective,initial_guess,constraints=cons)

    # Extraction des changements de rayons et calcul de position
    rayons=result.x

    y=((D1+rayons[0])**2-(D2+rayons[1])**2+y2**2)/(2*y2)
    x=((D1+rayons[0])**2-(D3+rayons[2])**2+x3**2)/(2*x3)
    z=np.sqrt(np.abs((D1+rayons[0])**2 - x**2 - y**2))

    return x,y,z
```

Résolution physique

```python
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt

#décalaration des constantes
gg = 9.81
rho =  1.292
S = 3.12e-3
Cx = 0.17
m = 0.0567
omega = 0

mu = 10e-4

t_max = 1.1
N_pas = 30000

delta =t_max/(N_pas -1)

#Schéma d'Euler explicite
def Euler(gg,rho,S,Cx,m,Vx0,Vy0,Vz0,N_pas,delta):
    Vx=[Vx0]
    Vy=[Vy0]
    Vz=[Vz0]

    for i in range(1,N_pas):
        Vx.append(Vx[i-1] -(0.5*(rho*S*Cx)/m*Vx[i-1]*(Vx[i-1]**2 + Vy[i-1]**2 + Vz[i-1]**2)**1/2)*delta)
        Vz.append(Vz[i-1]-(gg -0.5*(rho*S*Cx)/m*Vz[i-1]*(Vx[i-1]**2 + Vy[i-1]**2+Vz[i-1]**2)**1/2)*delta)
        Vy.append(Vy[i-1] -(0.5*(rho*S*Cx)/m*Vz[i-1]*(Vx[i-1]**2 + Vy[i-1]**2 + Vz[i-1]**2)**1/2)*delta)

    return Vx,Vy,Vz


Vx,Vy,Vz = Euler(gg,rho,S,Cx,m,9,7,5,N_pas,delta)
```

```python
#Intégration
def trapeze(delta, v, x0):
    x = np.zeros(len(Vx))
    x[0] = x0
    for i in range(1,len(x)):
        x[i] = x[i-1] + ((v[i]+v[i-1]))*delta/2
    return x

X = trapeze(delta, Vx,0)
Y = trapeze(delta,Vy,0)
Z = trapeze(delta,Vz,0)

# Tracé de la courbe 3D
ax = plt.axes(projection='3d')
plt.cla()
ax.plot3D(X, Y, Z, label='Courbe')
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.title('Exemple de résolution physique')
plt.tight_layout()
plt.show()
```

Vidéo

```python
import sys
import cv2
import os

# Read the video from specified path
cam = cv2.VideoCapture(r'C:\Users\heris\Desktop\TIPE\video.mp4')

try:

    # creating a folder named data
    if not os.path.exists('data'):
        os.makedirs('data')

# if not created then raise error
except OSError:
    print ('Error: Creating directory of data')

# frame
currentframe = 0

while(True):

    # reading from frame
    ret,frame = cam.read()

    if ret:
        # if video is still left continue creating images
        name = './data/frame' + str(currentframe) + '.jpg'
        print ('Creating...' + name)

        # writing the extracted images
        cv2.imwrite(name, frame)

        # increasing counter so that it will
        # show how many frames are created
        currentframe += 1
    else:
        break

# Release all space and windows once done
cam.release()
cv2.destroyAllWindows()
```

```python
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Traitement d'image
def haut(w,h):
    for y in range(h):
        for x in range(w):
            r,v,b=image.getpixel((x,y))
            if 150<r<=255 and 150<v<255 and 150<b<255:
                return x,y

def bas(w,h):
    for y in range(h-200,0,-1):
        for x in range(w):
            r,v,b=image.getpixel((x,y))
            if 150<r<=255 and 150<v<255 and 150<b<255:
                return x,y

l=[]
for i in range(20,81):
    print("image"+ str(i))
    image=Image.open('./images/ezgif-frame-0'+str(i)+'.jpg')
    w,h=image.size
    x1,y1=haut(w,h)
    x2,y2=bas(w,h)
    x3=(x1+x2)//2
    y3=(y1+y2)//2
    l.append((x3,y3))

xcmtab=[]
ycmtab=[]
for i in l:
    xcmtab.append((i[0])*0.05763)
    ycmtab.append((h-i[1])*0.05763)
```

Arduino

```
void loop() {
    if(DW1000NgRTLS::receiveFrame()){
        size_t recv_len = DW1000Ng::getReceivedDataLength();
        byte recv_data[recv_len];
        DW1000Ng::getReceivedData(recv_data, recv_len);


        if(recv_data[0] == BLINK) {
            DW1000NgRTLS::transmitRangingInitiation(&recv_data[2], tag_shortAddress);
            DW1000NgRTLS::waitForTransmission();

            RangeAcceptResult result = DW1000NgRTLS::anchorRangeAccept(NextActivity::RANGING_CONFIRM, next_anchor);
            if(!result.success) return;
            range_self = result.range;

            String rangeString = "Range "; rangeString += range_self; rangeString += " m";
            rangeString += "\t RX power: "; rangeString += DW1000Ng::getReceivePower(); rangeString += " dBm";
            Serial.println(rangeString);

        } else if(recv_data[9] == 0x60) {
            double range = static_cast<double>(DW1000NgUtils::bytesAsValue(&recv_data[10],2) / 1000.0);
            String rangeReportString = "Range from: "; rangeReportString += recv_data[7];
            rangeReportString += " = "; rangeReportString += range;
            Serial.println(rangeReportString);
        }
    }
}
```

Arduino

```cpp
#include <Arduino.h>
#include "DW1000Ng.hpp"
#include "DW1000NgConstants.hpp"
#include "DW1000NgRanging.hpp"
#include "DW1000NgConstants.hpp"
#include "DW1000NgRTLS.hpp"

namespace DW1000NgRanging {

    /* asymmetric two-way ranging (more computation intense, less error prone) */
    double computeRangeAsymmetric(
                                    uint64_t timePollSent,
                                    uint64_t timePollReceived,
                                    uint64_t timePollAckSent,
                                    uint64_t timePollAckReceived,
                                    uint64_t timeRangeSent,
                                    uint64_t timeRangeReceived
                                )
    {
        uint32_t timePollSent_32 = static_cast<uint32_t>(timePollSent);
        uint32_t timePollReceived_32 = static_cast<uint32_t>(timePollReceived);
        uint32_t timePollAckSent_32 = static_cast<uint32_t>(timePollAckSent);
        uint32_t timePollAckReceived_32 = static_cast<uint32_t>(timePollAckReceived);
        uint32_t timeRangeSent_32 = static_cast<uint32_t>(timeRangeSent);
        uint32_t timeRangeReceived_32 = static_cast<uint32_t>(timeRangeReceived);

        double round1 = static_cast<double>(timePollAckReceived_32 - timePollSent_32);
        double reply1 = static_cast<double>(timePollAckSent_32 - timePollReceived_32);
        double round2 = static_cast<double>(timeRangeReceived_32 - timePollAckSent_32);
        double reply2 = static_cast<double>(timeRangeSent_32 - timePollAckReceived_32);

        int64_t tof_uwb = static_cast<int64_t>((round1 * round2 - reply1 * reply2) / (round1 + round2 + reply1 + reply2));
        double distance = tof_uwb * DISTANCE_OF_RADIO;

        return distance;
    }
```