Analyse du temps de réaction humain

Thème TIPE 2024 : Jeux, Sports

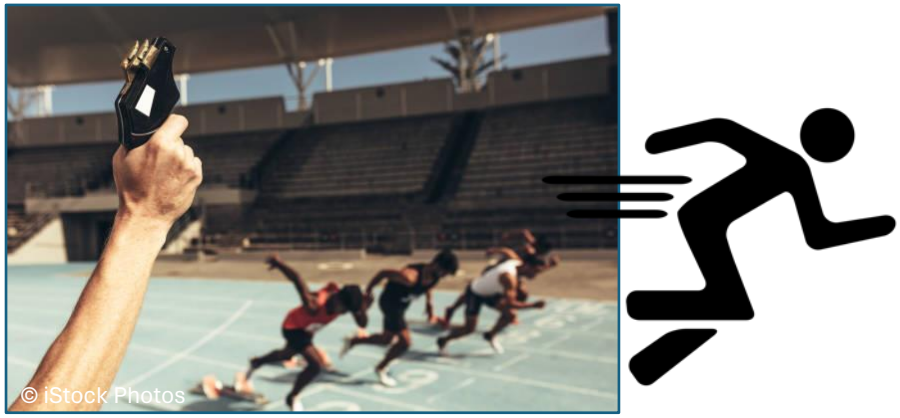


Figure 1.1 – Starter de sprint en athlétisme

Figure 1.2 – Compétition de e-sport

Côme Ciavarella – N°32026

CPGE PT

Figure 1.3 – Joueurs de tennis de table

Améliorer les performances sportives
(liées aux temps de réaction)

Figure 1.4 – Chute de cyclistes sur route

Prévenir les évènements redoutés
(blessures, collisions, accidents)

Problématique

Est-il pertinent de vouloir optimiser le temps de réaction humain afin d'améliorer les performances sportives ?

Sommaire TIPE

Projet d'ingénierie

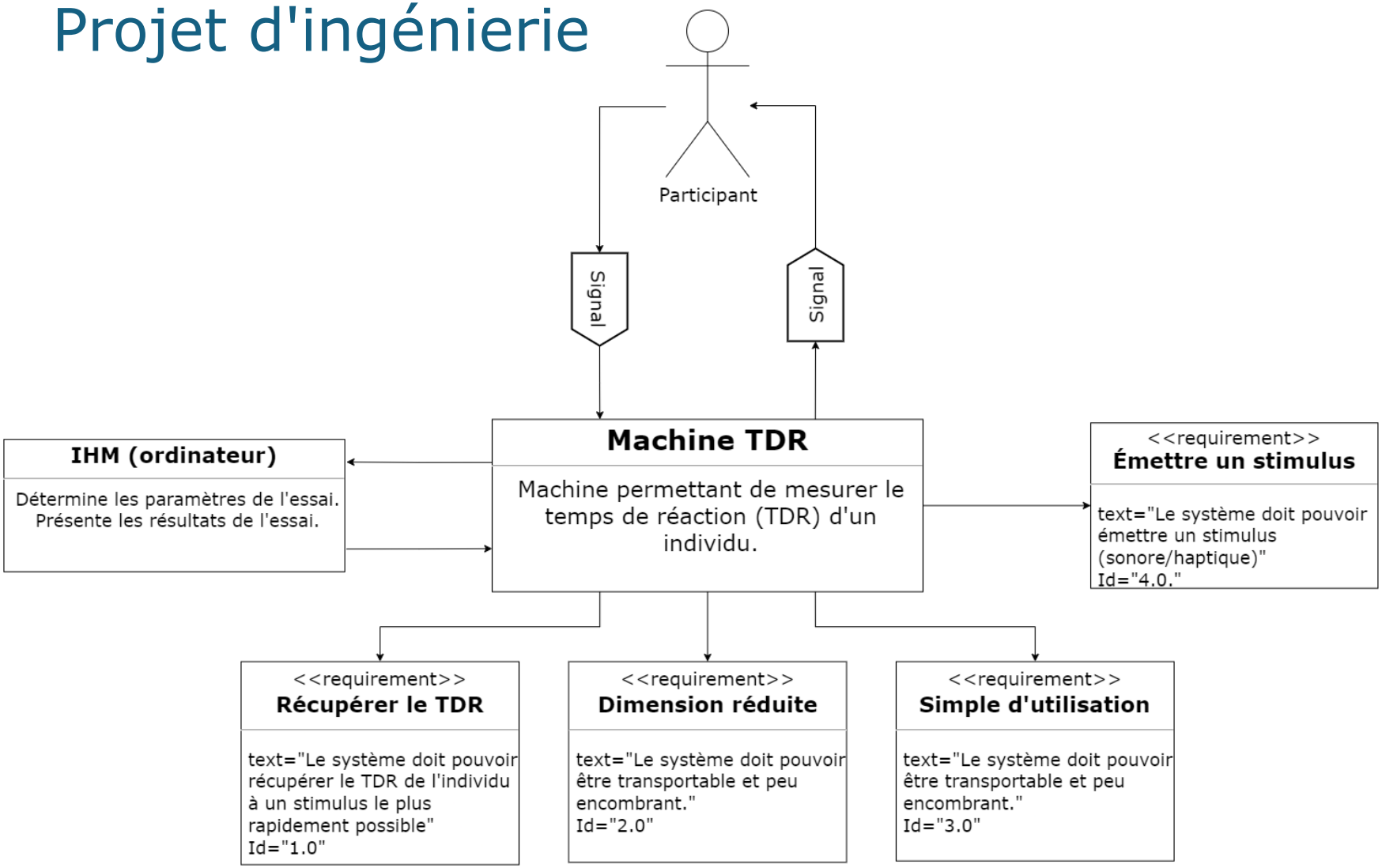


Figure 2.1 – Diagramme de contraintes SysML

Projet d'ingénierie

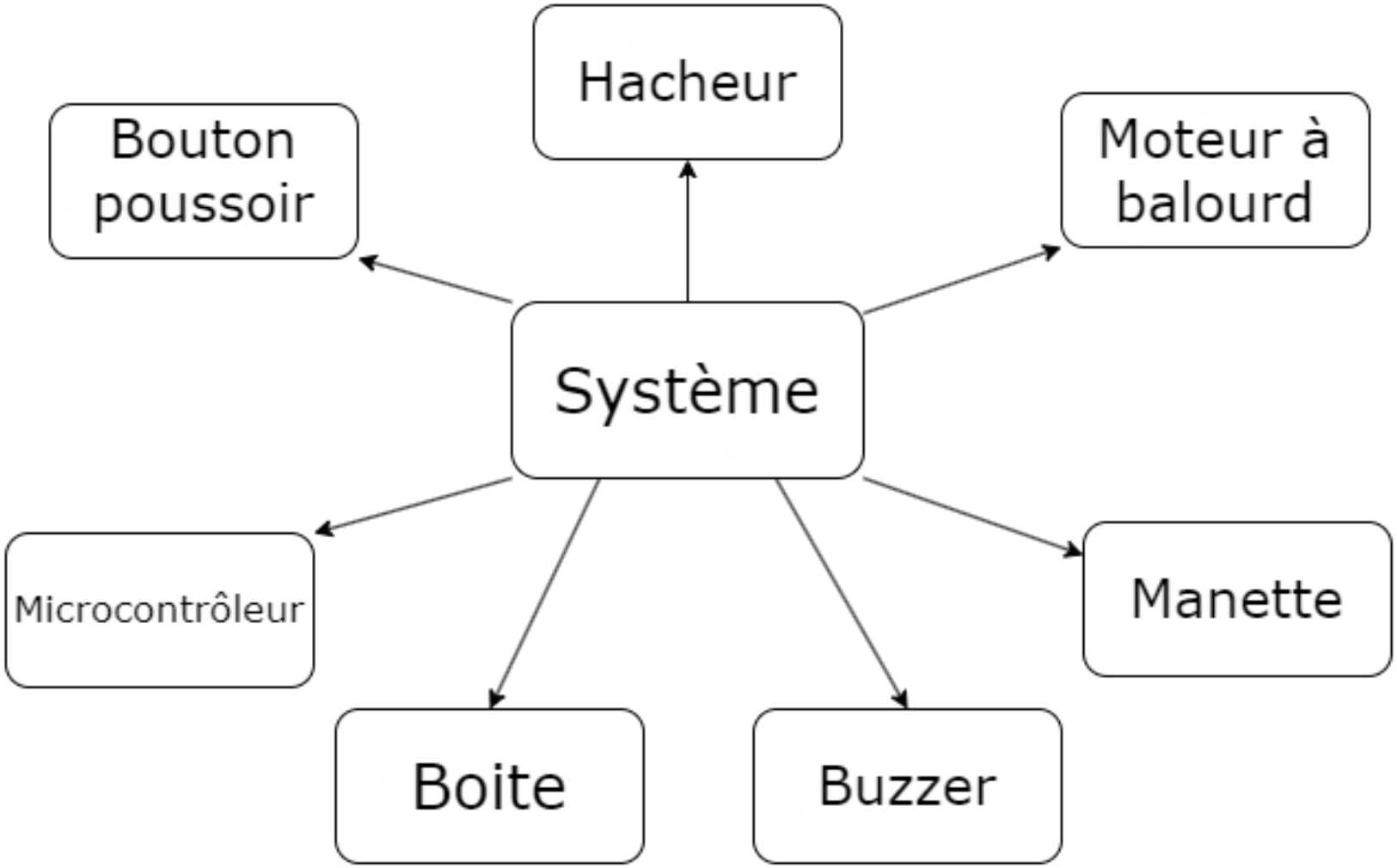
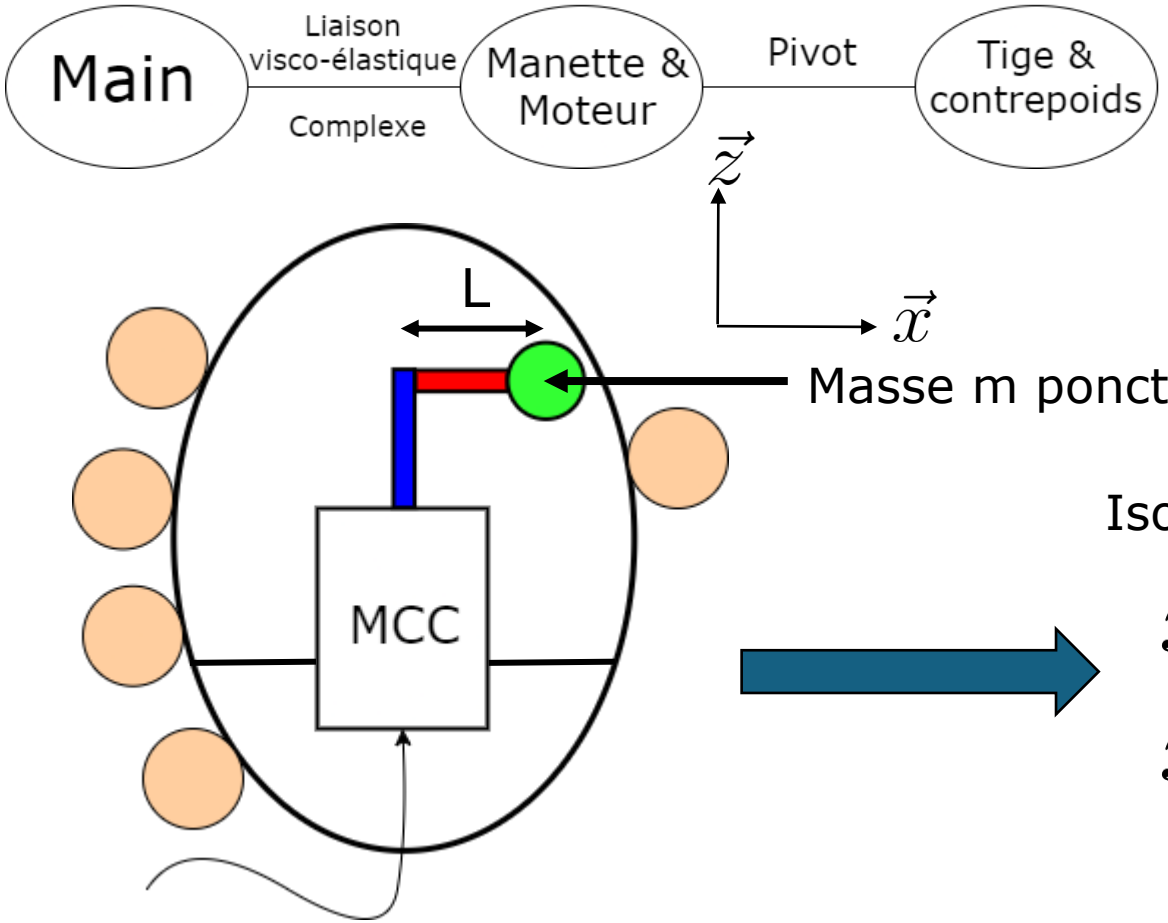


Figure 2.2 – Diagramme de blocs

Projet d'ingénierie



Hypothèses :
- Solide rigide
- Masse des tiges négligées

Isolement de la masse et PFD

$$\vec{x} : X = mL\ddot{\theta}$$
$$\vec{x} : X = mL\dot{\theta}^2$$

Figure 2.3 – Schéma de principe de la manette vibrante

Hardware

Microcontrôleur ESP 32

- Compatible python (V3.4)
- Temps de réponse rapide
- Coût réduit (5 euros)

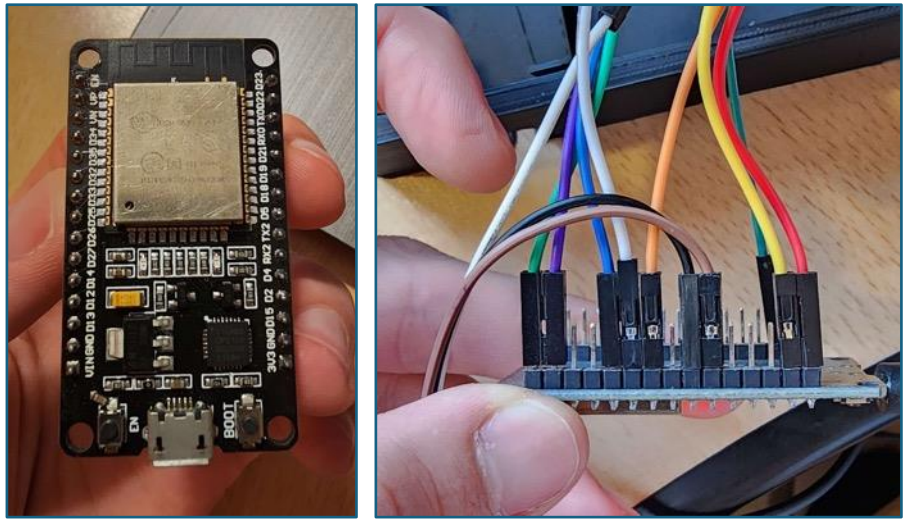


Figure 2.4 – Microcontrôleur ESP 32

Hardware

Moteur à balourd

- Courant continu
- Vibrations variables (non équilibré statiquement et dynamiquement)
- Coût : s/o, recyclage

Figure 2.5 – Moteur à balourd machine

Inspiration de dispositifs existants

Figure 2.6 – Moteur à balourd de manette de jeux

Hardware

Hacheur L298N

- 4 quadrants
- Limites : 35 V / 4 A max.
- Coût réduit (2 euros)

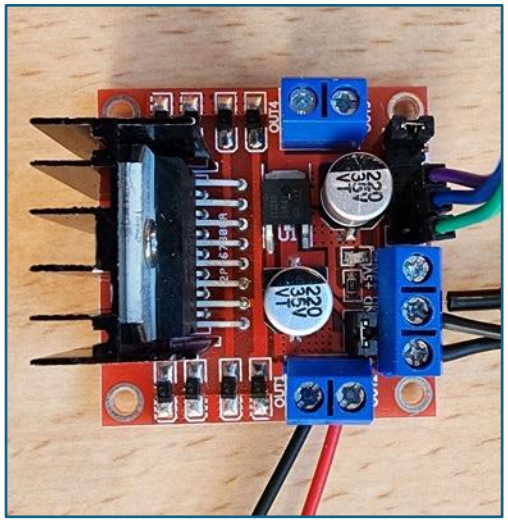


Figure 2.7 – Hacheur 4 quadrants machine

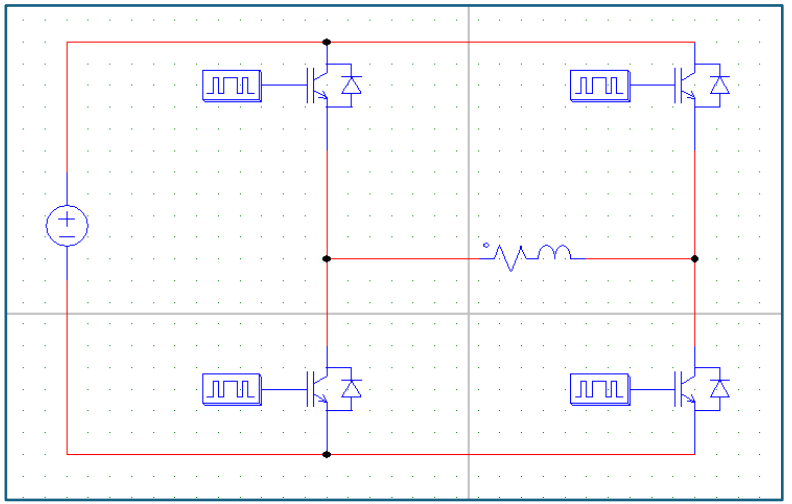


Figure 2.8 – Schéma électrique hacheur 4 quadrants

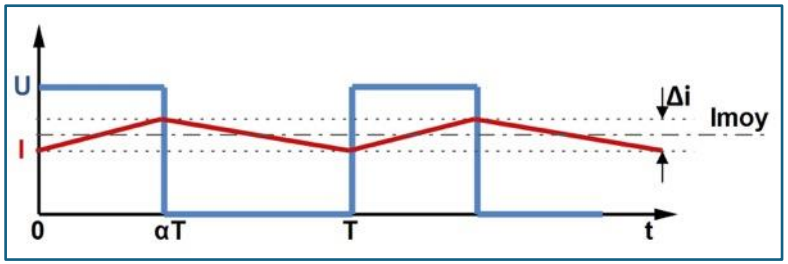


Figure 2.9 – Signaux traversants le moteur

Hardware

Buzzer

- Connexion plug & play
- Coût : s/o, recyclage

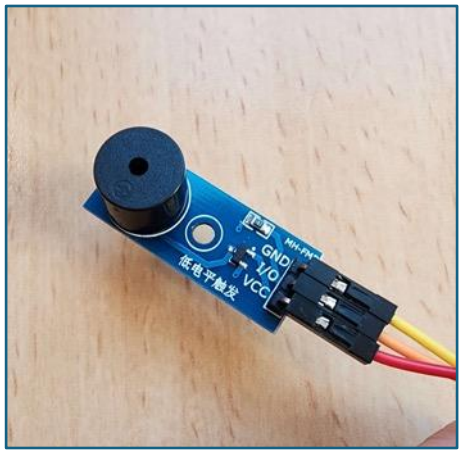


Figure 2.10 – Buzzer machine

Bouton poussoir

- Course faible
- Rebond limité
- Coût réduit (50 cents)

Figure 2.11 – Bouton poussoir machine

Conception 3D

Modélisation Solidworks

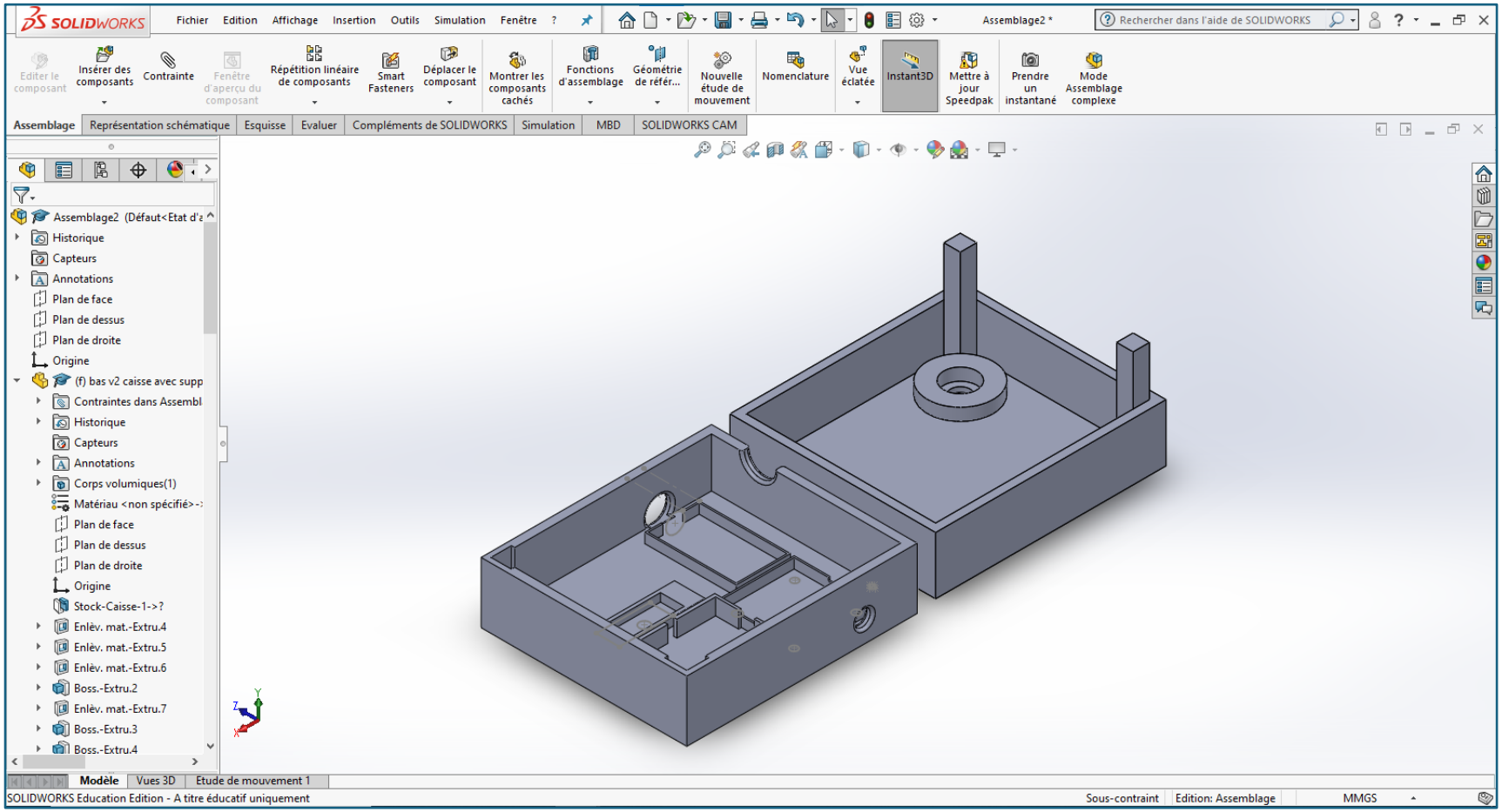


Figure 2.12 – Modèles Solidworks

Impression 3D

Exportation des modèles Solidworks vers Cura

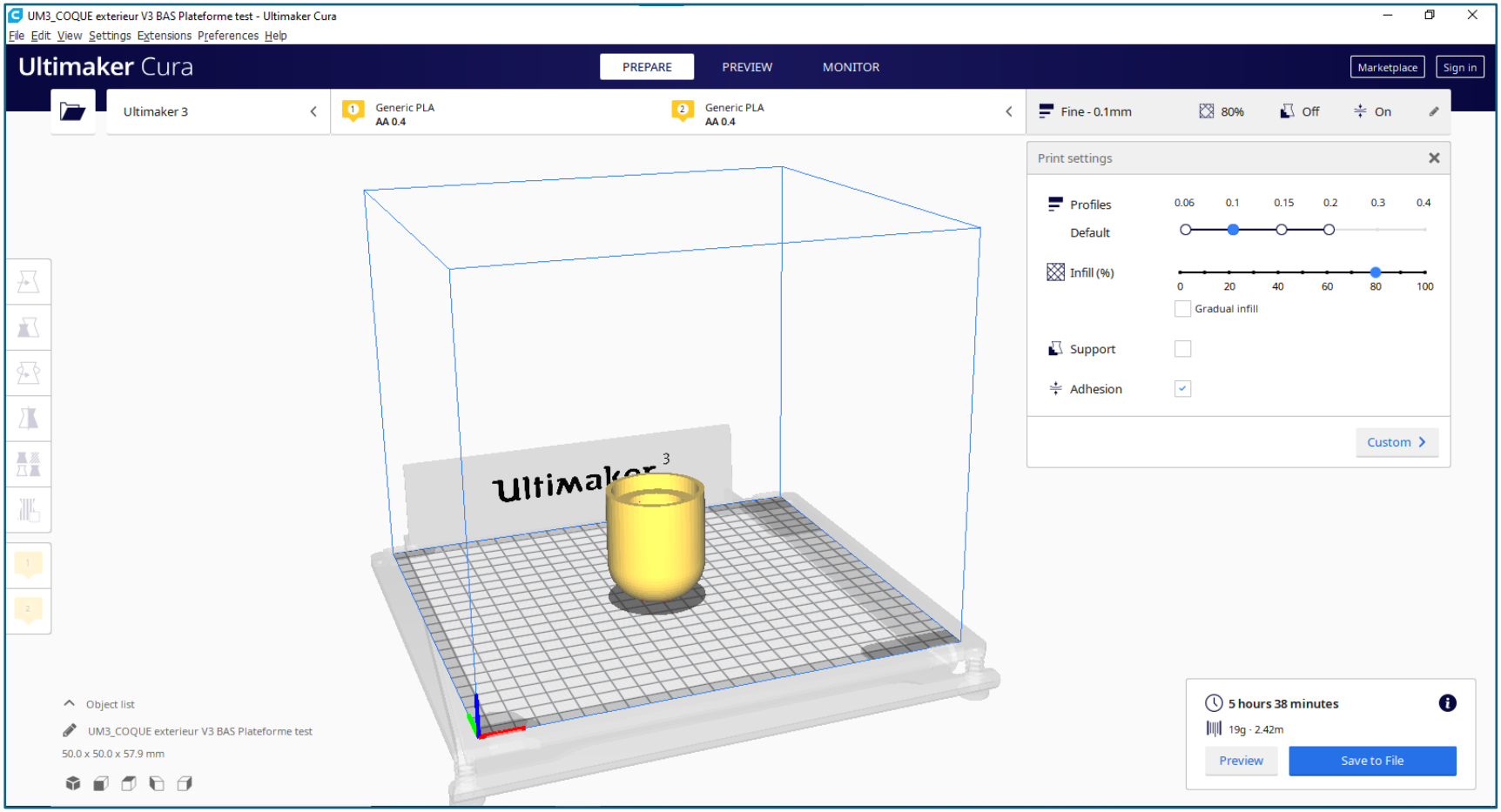


Figure 2.13 – Modèle 3D sur Cura

Conception de la manette

Assemblage de la manette vibrante

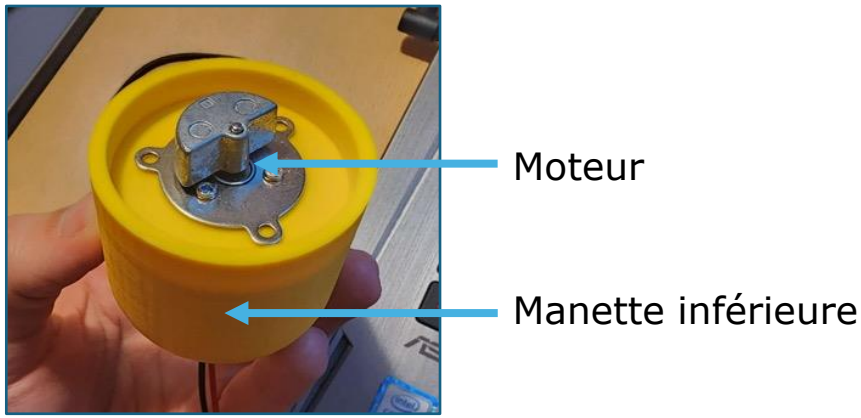


Figure 2.14 – Montage moteur

Figure 2.15 – Manette en PLA assemblée

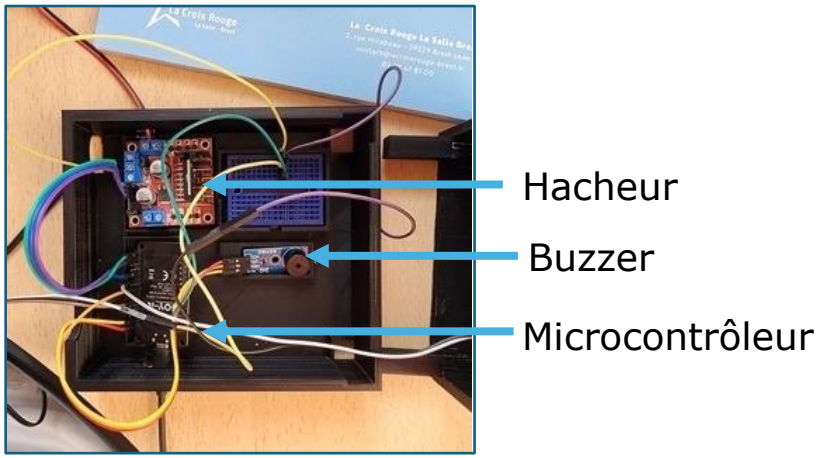


Figure 2.16 – Boitier en PLA unité centrale

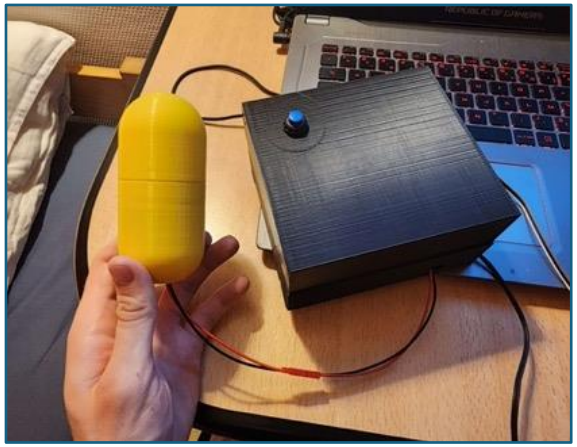


Figure 2.17 – Machine assemblée

Software

Programmation du microcontrôleur

- IDE uPyCraft v1.1 : Environnement de développement Python pour ESP 32
- Interface USB (PC / ESP 32)

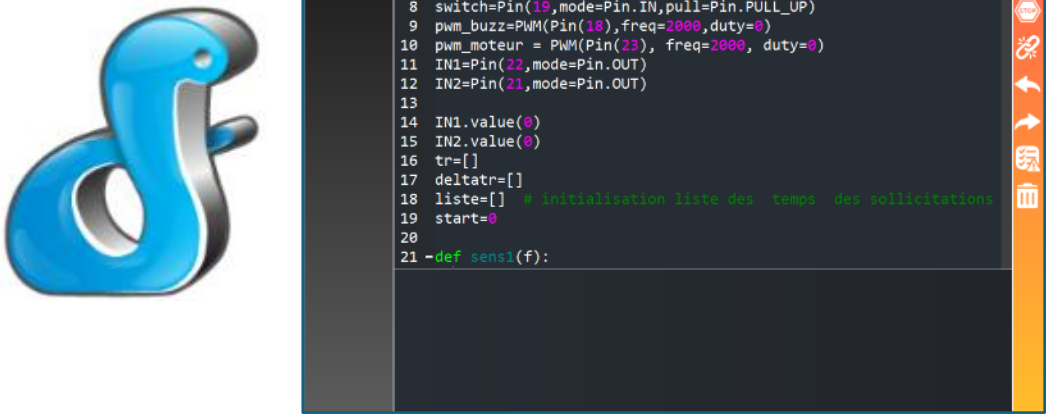


Figure 2.18 – Interface IDE uPyCraft

Software IHM

IHM de commande du microcontrôleur sous PyQT 4

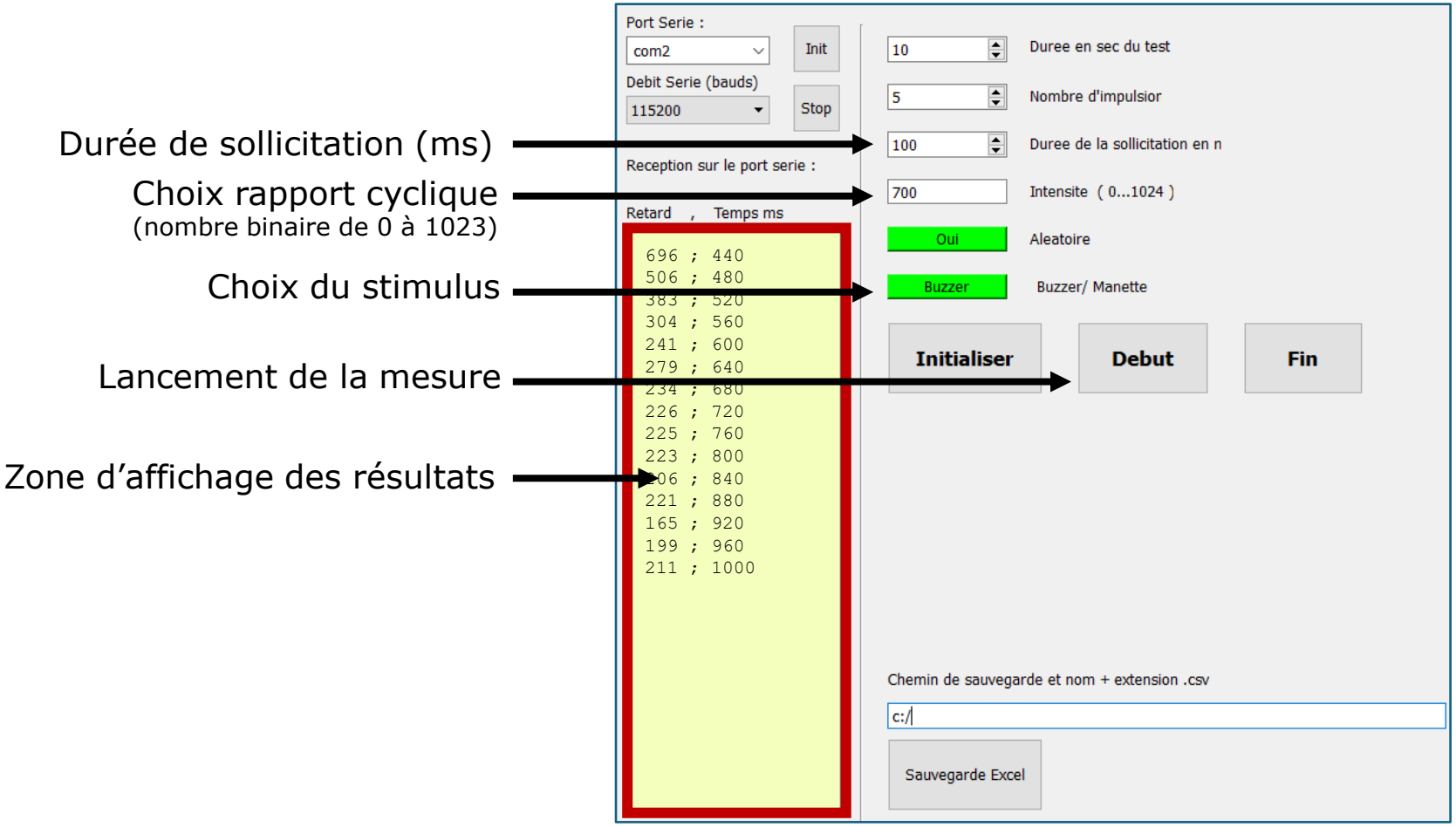


Figure 2.19 – IHM de commande du microcontrôleur

Préparation des expériences

- Participants : panel de 6 personnes
- Profils : 4 types différents
  - Sportif (1 individu)
  - Gamer (2 individus)
  - Sportif et Gamer (2 individus)
  - Non sportif et non gamer (1 individu)
- Durée d'étude : 09 jours
- Conditions de réalisation identiques (lieu, environnement, matériels)

Figure 3.1 – Configuration de la machine

Protocole Expérience n°1

Mesure du TDR à un stimulus sonore et haptique

- Instructions participants, test à blanc
- Prise de 5 mesures pour chaque stimulus (sonore, haptique)

Figure 3.2 – Participante Expérience n°1

Protocole Expérience n°2

Mesure du TDR à des stimuli d'intensités croissantes

- Instructions participants, test à blanc
- Prise de 15 mesures de rapport cyclique croissant par pas de 40 de 440 à 1000 (nombres binaires)

Figure 3.3 – Participant Expérience n°2

Traitement des données

Expérience n°1

- Séparation des résultats de chaque test (sonore, haptique)
- Calcul moyenne, incertitude-type des TDR (groupée, individuelle)

Expérience n°2

- Regroupement des résultats de chaque test
- Calcul moyenne, incertitude-type des TDR (groupée, individuelle)
- IA algorithme descente de gradient

Analyse Expérience n°1

Profils : ensemble des 4 profils

- Progression journalière moyenne : **6,9** ms (**3,2** % du TDR initial)
- Évolution totale : **55,2** ms (**25,9** % du TDR initial)
- Evolution écart type : **36,1** ms (**66,9** % de l'écart type initial moyen)

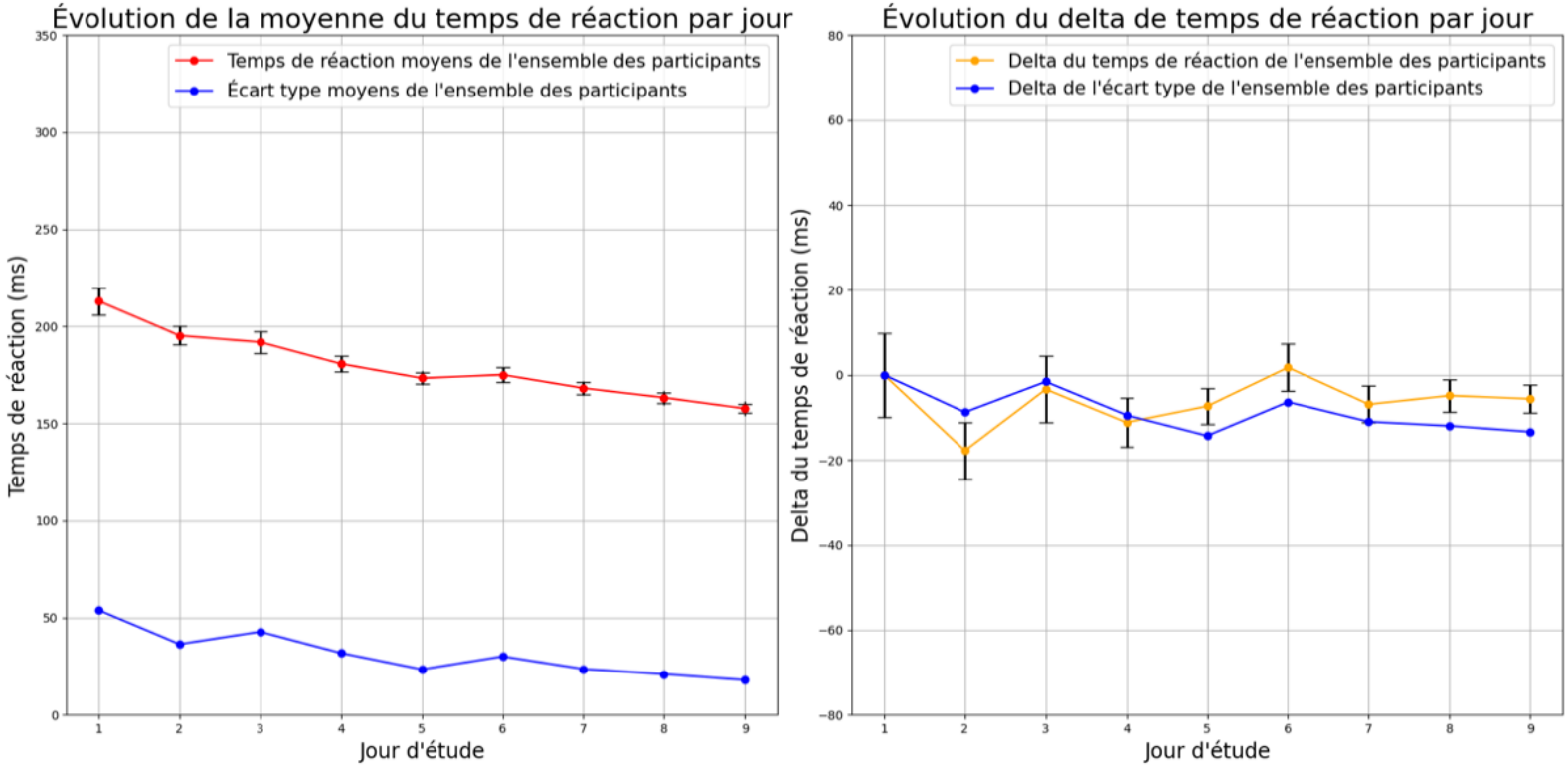


Figure 3.4 – Evolutions des résultats Expérience n°1 (tous profils)

Analyse Expérience n°1 par profil

Sportif, gamer

- Progression journalière :

3,4 ms (**2,0** % du TDR initial moyen)

- Évolution totale moyenne :

27,0 ms (**15,7** % du TDR initial moyen)

- Evolution écart type :

21,6 ms (**65,3** % de l'écart type initial moyen)

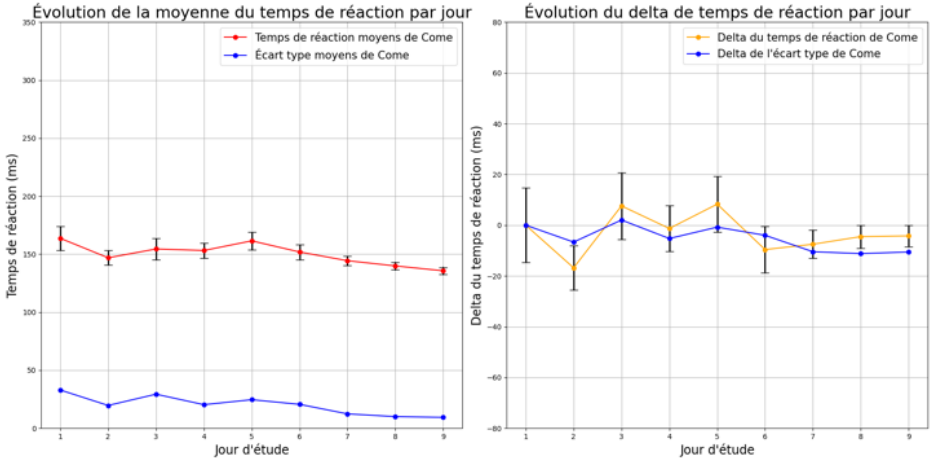


Figure 3.5 – Résultats Expérience n°1 « sportif, gamer »

Non sportif, non gamer

- Progression journalière :

12,8 ms (**4,9** % du TDR initial)

- Évolution totale moyenne :

102,4 ms (**39,2** % du TDR initial)

- Evolution écart type :

58,6 ms (**76,5** % de l'écart type initial moyen)

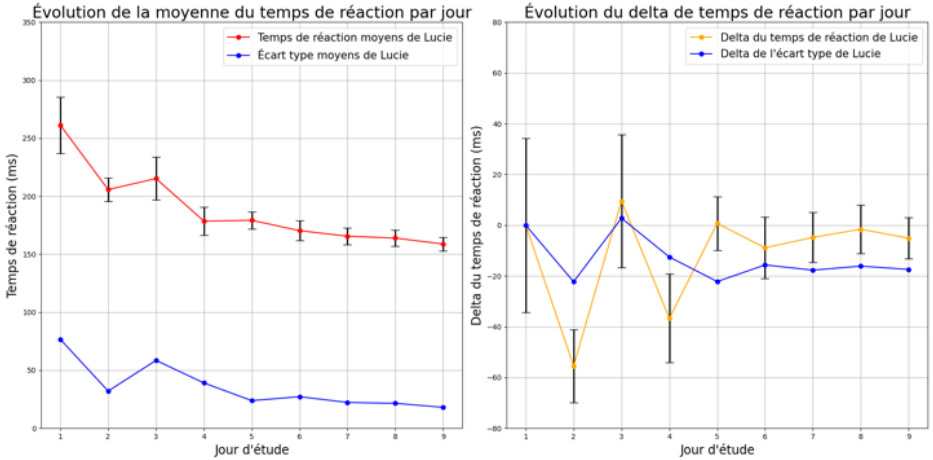


Figure 3.6 – Résultats Expérience n°1 « non sportif, non gamer »

Analyse Expérience n°1 par profil

Sportif

- Progression journalière :

6,0 ms (**2,9** % du TDR initial moyen)

- Évolution totale moyenne:

48,2 ms (**23,2** % du TDR initial moyen)

- Evolution écart type :

18,1 ms (**65,8** % de l'écart type initial moyen)

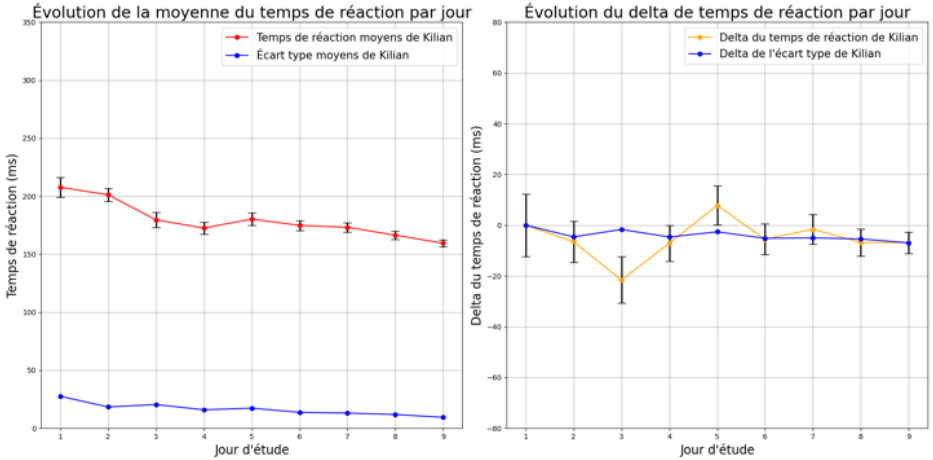


Figure 3.7 – Résultats Expérience n°1 « sportif »

Gamer

- Progression journalière :

8,0 ms (**3,4** % du TDR initial)

- Évolution totale moyenne :

63,5 ms (**27,0** % du TDR initial)

- Evolution écart type :

15,3 ms (**55,6** % de l'écart type initial moyen)

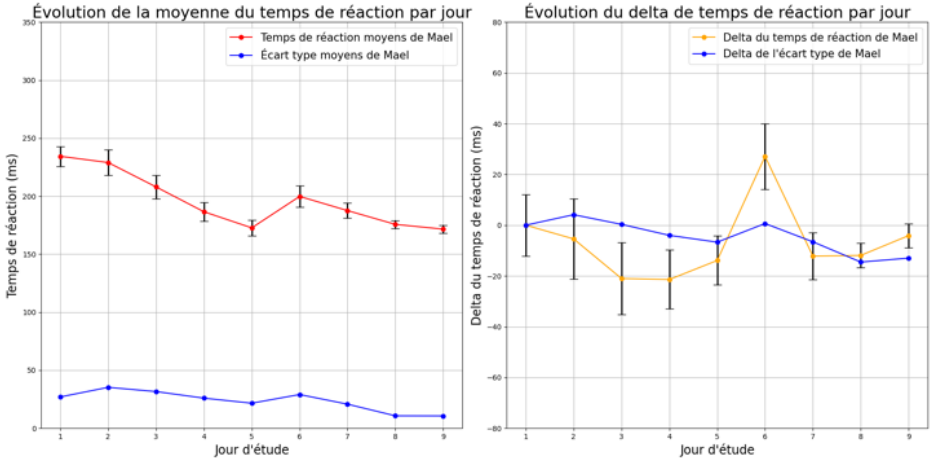
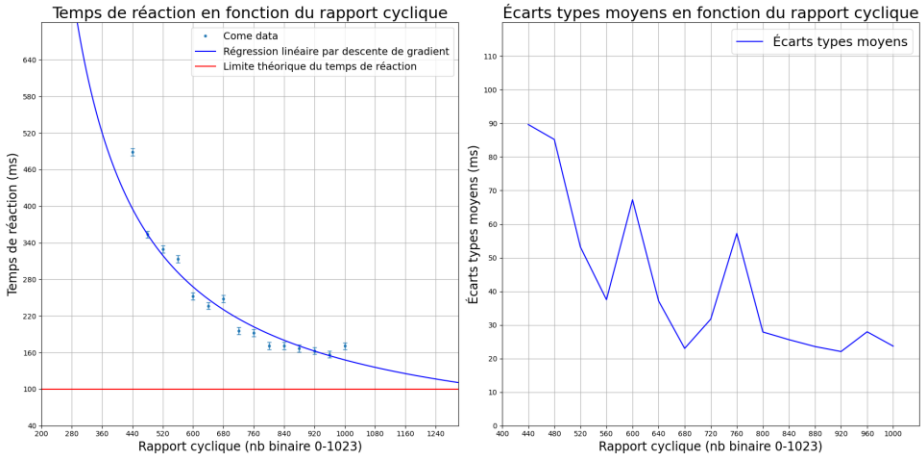


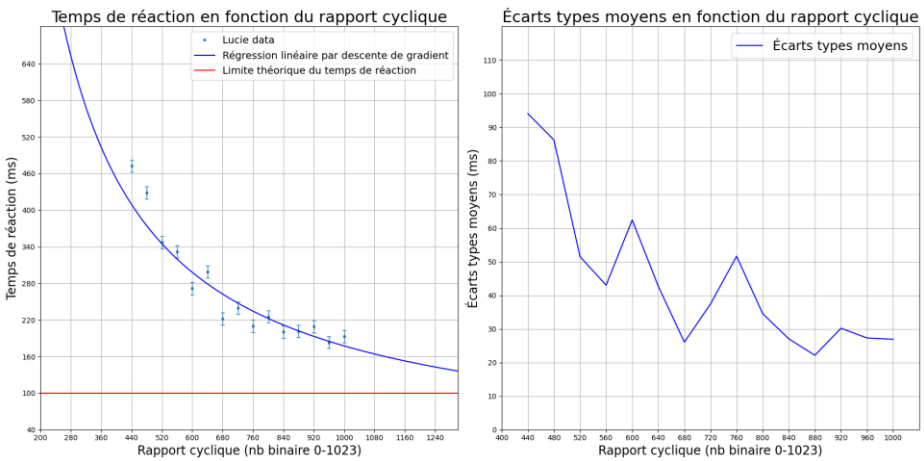
Figure 3.8 – Résultats Expérience n°1 « gamer »

Analyse Expérience n°2 par profil

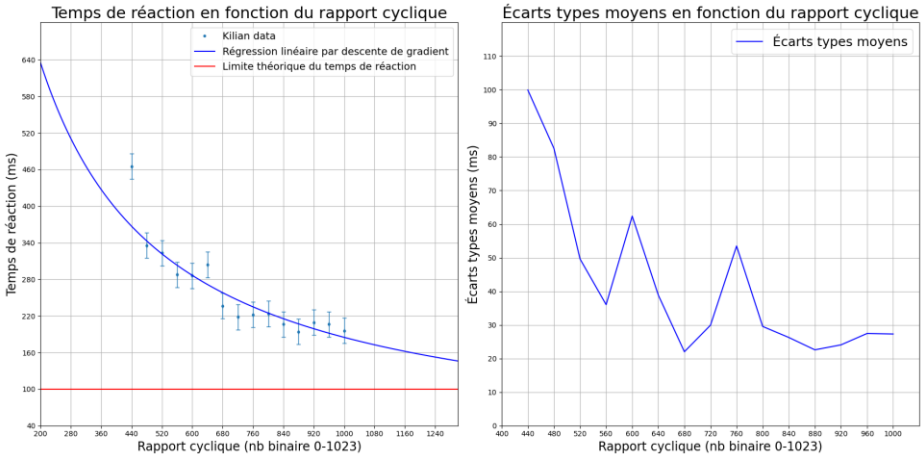
Sportif, gamer



Non sportif, non gamer



Sportif



Gamer

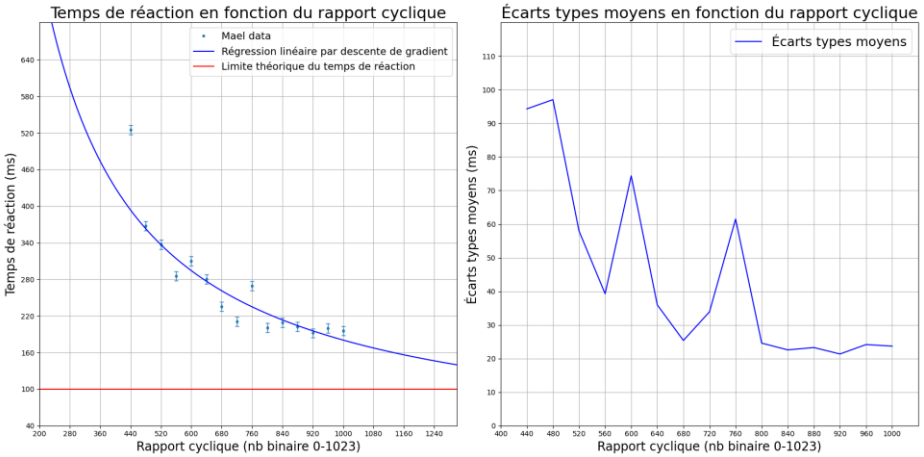


Figure 3.9 – Résultats Expérience n°2 « sportif, gamer » et « sportif »

Figure 3.10 – Résultats Expérience n°2 « non sportif, non gamer » et « gamer »

Analyse Expérience n°2

Profils : ensemble des 4 profils

- Equation de la courbe (descente de gradient) : $y = \frac{1}{6,01 \times 10^{-6}x - 4,84 \times 10^{-5}}$
- Progression totale : **293,1** ms (**61,0** % du TDR initial)
- Progression totale écart type : **64,3** ms (**69,2** % de l'écart type initial moyen)

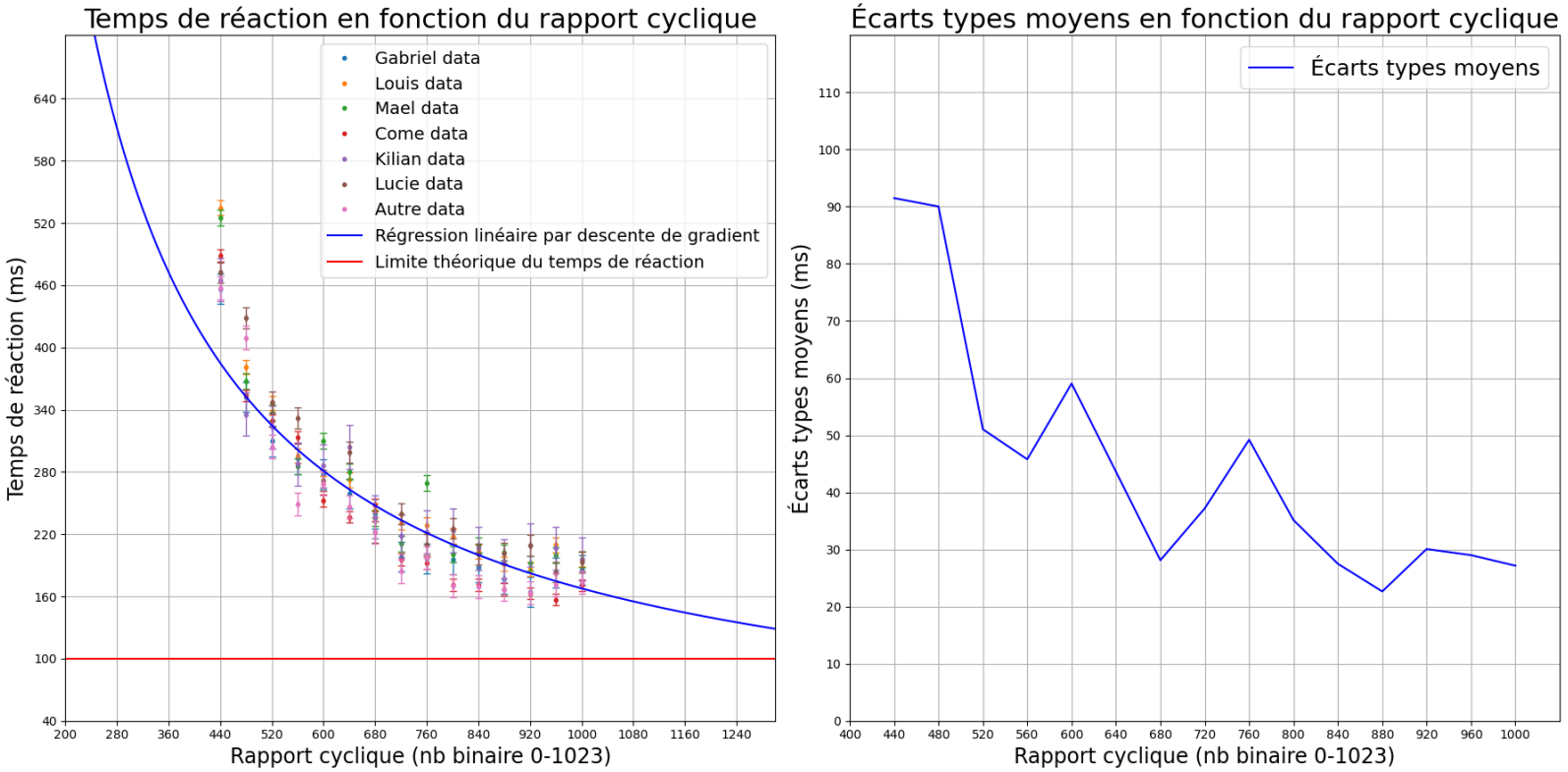


Figure 3.11 – Evolutions des résultats Expérience n°2 (tous profils)

Conclusion

Est-il pertinent de vouloir optimiser le temps de réaction humain afin d'améliorer les performances sportives ?

- Entrainement **pertinent** :
  - Gain de **27,0** ms au minimum en 9 jours (4 fois plus faible que pour les non sportifs, non gamers)
  - Gain de **36,1** ms d'écart-type en moyenne

- Privilégier les stimuli de **haute intensité** :
  - **293,1** ms de moins que l'intensité la plus faible en moyenne
  - **64,3** ms d'écart-type en moins en moyenne

- Corroboration de la limite théorique de l'homme (**100** ms)

Perspectives de développement

- Amélioration du protocole:
  - Prendre en compte plus de paramètres (sommeil, heures de test)
    - Sommeil
    - Heures de mesures (matin/midi/soir)
- Avoir les données d'un sportif de haut niveau
- Personnaliser les stimuli du sport en question

Figure 4.1 – Entrainement du temps de réaction d'un pilote de F1

Apports personnels

- Conception Solidworks
- Impression 3D
- Commande d’un microcontrôleur ESP 32
- Démarche scientifique

Table des illustrations

Table des illustrations

Table des illustrations

Table des annexes

Annexe I - Sources images

Figure 1.1 – Starter de sprint en athlétisme : : istockphoto.com

Figure 1.2 – Compétition de e-sport : riotgames.com

Figure 1.3 – Joueurs de tennis de table : istockphoto.com

Figure 1.4 – Chute de coureurs cyclistes sur route : : istockphoto.com

Figure 2.6 – Moteur à balourd de manette de jeux vidéo : buzzygame.fr

Figure 2.8 – Schéma électrique hacheur 4 quadrants : forums.futura-sciences.com

Figure 2.9 – Signaux traversants le moteur : naytech.blogspot.com

Figure 4.1 – Entrainement du temps de réaction d'un pilote de F1 : istockphoto.com

Figure VI.1 – Représentation de l'allure d'une décroissance : étude H.Piéron (1932)

Annexe II – 1/13 Programme python traitement des données

```python
import csv
import matplotlib.pyplot as plt
import numpy as np
import json
import os
import fnmatch
resultat_indiv={}
res_gradient_vib={}
dico_intens_vide={}
rapport_cyclique = np.linspace( start: 440, stop: 1000, num: 15)
moy_ec_intens={int(intens) : [] for intens in rapport_cyclique}



2 usages
def compter_csv_commencant_par(dossier, prefixe):
    nombre_fichiers = 0
    for fichier in os.listdir(dossier):
        if fnmatch.fnmatch(fichier, pat: f"{prefixe}*.csv"):
            nombre_fichiers += 1
    return nombre_fichiers
```

Annexe II – 2/13 Programme python traitement des données

```
def ajout_dico_evol(prenom,numtestdep=1,numtestarr=0): #
    """
    Ajoute les test allant de numtestdep à numtestarr (inclus) de prenom au dico resultat_indiv
    """
    if numtestarr==0:
        numtestarr = compter_csv_commencant_par("TIPE/CODE/DONNEES/SESSION TEST/ ".strip() + prenom.lower(),prenom.lower() +'test')
    resultat_indiv[prenom.lower()]=[0,[],[]]
    #Nom des individus : nb de tests, liste des temps de réac au buzzer, liste des temps au vibreur. Toute les 5 données on a un jour (Vibreur et buzzer sont les mêmes.
    for i in range(numtestdep,numtestarr+1,1):
        resultat_indiv[prenom.lower()][0]+=1
        with open ("TIPE/CODE/DONNEES/SESSION TEST/ ".strip()+prenom.lower()+"/ ".strip() + prenom.lower() +'test'+str(i)+'.csv',newline='') as fichier_csv:
            lecteur= csv.reader(fichier_csv, delimiter=';')
            compteur=0
            lecteur.__next__() #passe l'en-tête
            for ligne in lecteur:
                if any(field.strip() for field in ligne):
                    try:
                        compteur+=1
                        if compteur<=5:
                            resultat_indiv[prenom.lower()][1].append(int(ligne[0]))
                        else:
                            resultat_indiv[prenom.lower()][2].append(int(ligne[0]))
                    except:
                        pass
```

Annexe II – 3/13 Programme python traitement des données

```python
def ajout_intens(prenom, numtestdep=1 ,numtestarr=0):
    if numtestarr == 0:
        numtestarr = compter_csv_commencant_par("TIPE/CODE/DONNEES/SESSION TEST/ ".strip() + prenom.lower(),prenom.lower() +"intens")
    dico_intens_vide = {int(i): [] for i in rapport_cyclique}
    res_gradient_vib[prenom.lower()] = dico_intens_vide.copy()
    # Nom des individus : nb de tests, liste des temps de réac au buzzer, liste des temps au vibreur. Toute les 5 données on a un jour (Vibreur et buzzer sont les mêmes.
    for i in range(numtestdep, numtestarr + 1, 1):
        with open("TIPE/CODE/DONNEES/SESSION TEST/ ".strip()+prenom.lower()+"/ ".strip() + prenom.lower() +'intens'+str(i)+'.csv',newline='') as fichier_csv:
            lecteur = csv.reader(fichier_csv, delimiter=';')
            next(lecteur)
            for ligne in lecteur:
                if any(field.strip() for field in ligne):
                    try:
                        res_gradient_vib[prenom.lower()][int(ligne[1])].append(int(ligne[0]))
                    except(KeyError, ValueError):
                        pass
```

Annexe II – 4/13 Programme python traitement des données

```
def analyse_indiv_rapid(prenom,donnees,type=''):
    """
        Calcule la moyenne, variance, écart-type des données
        return([[moy_buzzer,moy_vibreur,moy_tot],[var_buzzer,var_vibreur,var_tot],[ectype_buzzer,ectype_vibreur,ectype_tot]])

    :param prenom: 'tout' ou 'mael (par exemple)
    :param donnees: 'intens' ou quelconque (Intens pour le gradient d'intensité)
    :param type: 'moy','var','ectype'
    :return:
    """
    if donnees=='intens':
        if prenom.lower()=='tout':
            donnes_dico={}
            for nom in res_gradient_vib:
                if type.lower()=='moy' or type.lower()=='moyenne':
                    donnes_dico[nom]={key: round(np.mean(values),2) for key, values in res_gradient_vib[nom].items()}
                elif type.lower() == 'var' or type.lower() == 'variance':
                    donnes_dico[nom] = {key: round(np.var(values),2) for key, values in res_gradient_vib[nom].items()}
                elif type.lower() == 'ectype' or type.lower() == 'ecart type':
                    donnes_dico[nom] = {key: round(np.std(values),2) for key, values in res_gradient_vib[nom].items()}
        return( donnes_dico )
```

Annexe II – 5/13 Programme python traitement des données

```
else:
    if prenom.lower() == 'tout':
        res_global=[[],[]]
        for nom in resultat_indiv:
            res_global[0].extend(resultat_indiv[nom][1][:])
            res_global[1].extend(resultat_indiv[nom][2][:])
        if type.lower()=='moy' or type.lower()=='moyenne':
            moy_buzzer = np.mean(res_global[0])
            moy_vibreur = np.mean(res_global[1])
            moy_tot = np.mean(resultat_indiv[prenom][1] + res_global[1])
            return([moy_buzzer,moy_vibreur,moy_tot])
        elif type.lower()=='var' or type.lower()=='variance':
            var_buzzer = np.var(res_global[0])
            var_vibreur = np.var(res_global[1])
            var_tot = np.var(res_global[0] + res_global[1])
            return([var_buzzer,var_vibreur,var_tot])
        elif type.lower() == 'ectype' or type.lower() == 'ecart type':
            ectype_buzzer = np.std(res_global[0])
            ectype_vibreur = np.std(res_global[1])
            ectype_tot = np.std(res_global[0] + res_global[1])
            return([ectype_buzzer,ectype_vibreur,ectype_tot])
    else:
        if type.lower()=='moy' or type.lower()=='moyenne':
            moy_buzzer = np.mean(resultat_indiv[prenom][1])
            moy_vibreur = np.mean(resultat_indiv[prenom][1])
            moy_tot = np.mean(resultat_indiv[prenom][1] + resultat_indiv[prenom][1])
            return([moy_buzzer,moy_vibreur,moy_tot])
        elif type.lower()=='var' or type.lower()=='variance':
            var_buzzer = np.var(resultat_indiv[prenom][1])
            var_vibreur = np.var(resultat_indiv[prenom][1])
            var_tot = np.var(resultat_indiv[prenom][1] + resultat_indiv[prenom][1])
            return([var_buzzer,var_vibreur,var_tot])
        elif type.lower() == 'ectype' or type.lower() == 'ecart type':
            ectype_buzzer = np.std(resultat_indiv[prenom][1])
            ectype_vibreur = np.std(resultat_indiv[prenom][1])
            ectype_tot = np.std(resultat_indiv[prenom][1] + resultat_indiv[prenom][1])
            return([ectype_buzzer,ectype_vibreur,ectype_tot])
```

Annexe II – 6/13 Programme python traitement des données

```
def graph_evojour(prenom='TOUT',type='TOUT'):
    """
    prenom = Nom de l'individu ou TOUT si pour tout les individus
    type = Buzzer, Vibreur, Tout
    Trace le graphique de la progression de chaque individu de façon individuelle par jour (Avec écart type)
    Et renvoie la moyenne de l'évolution par jour de prenom
    """
    liste_donnees = [[], []]  # indice 0 : Temps du buzzer ; Indice 1 : Temps du vibreur
    liste_moyennejour = [] #liste des moyennes par jour
    tdrevol_jour = [] #évolution des moyennes par jour
    liste_ectypejour=[]
    eccevol_jour=[]
    fig, axs = plt.subplots( nrows: 1,  ncols: 2, figsize=(18, 9))
    nb_participant=6
    if prenom.upper() != 'TOUT':
        nb_jour = len(resultat_indiv[prenom.lower()][1])//5
        nb_participant=1
    else:
        nb_jour = len(resultat_indiv[list(resultat_indiv.keys())[0]][1])//5
    liste_jour = np.arange(nb_jour) + 1
    if prenom.upper() != 'TOUT':
        liste_donnees[0]+=resultat_indiv[prenom.lower()][1]
        liste_donnees[1]+=resultat_indiv[prenom.lower()][2]
    else:
        for i in range(nb_jour):
            for nom in resultat_indiv:
                liste_donnees[0]+=resultat_indiv[nom][1][5*i:5*i+5]
                liste_donnees[1]+=resultat_indiv[nom][2][5*i:5*i+5]
    if type.upper()=='TOUT' or type.upper()=='2' or type.upper()=='buzzer + vibreur' or type.upper()=='vibreur + buzzer':
        for i in range(nb_jour):
            moy_jouri = np.mean(liste_donnees[0][5*i*nb_participant:(5*i+5)*nb_participant] + liste_donnees[1][5*i*nb_participant:(5*i+5)*nb_participant])
            ec_jouri = np.std(liste_donnees[0][5*i*nb_participant:(5*i+5)*nb_participant] + liste_donnees[1][5*i*nb_participant:(5*i+5)*nb_participant])
            liste_moyennejour.append(moy_jouri) #commence au jour 0 (1er jour de test)
            tdrevol_jour.append(liste_moyennejour[i]-liste_moyennejour[i-1])
            liste_ectypejour.append(ec_jouri)
            eccevol_jour.append(liste_ectypejour[i]-np.mean(liste_ectypejour))
            axs[0].errorbar(i+1, liste_moyennejour[i], yerr=liste_ectypejour[i]/np.sqrt(5*nb_participant*2), xerr=0, linewidth=2,capsize=6,c='black')
            axs[1].errorbar(i + 1, tdrevol_jour[i], yerr=liste_ectypejour[i] / np.sqrt(5 * nb_participant), xerr=0,linewidth=2, capsize=6, c='black')
```

Annexe II – 7/13 Programme python traitement des données

```
elif type.upper()=='BUZZER' or type.upper()=='SON' or type.upper()=='BRUIT' or type.upper()=='BIP':
    for i in range(nb_jour):
        moy_jouri = np.mean(liste_donnees[0][5*i*nb_participant:(5*i+5)*nb_participant])
        ec_jouri = np.std(liste_donnees[0][5*i*nb_participant:(5*i+5)*nb_participant])
        liste_moyennejour.append(moy_jouri)  # commence au jour 0 (1er jour de test)
        tdrevol_jour.append(liste_moyennejour[i] - liste_moyennejour[i - 1])
        liste_ectypejour.append(ec_jouri)
        eccevol_jour.append(liste_ectypejour[i] - np.mean(liste_ectypejour))
        axs[0].errorbar(i+1, liste_moyennejour[i], yerr=liste_ectypejour[i] / np.sqrt(5 * nb_participant), xerr=0,fmt='o', linewidth=2, capsize=6,c='r')
        axs[1].errorbar(i + 1, tdrevol_jour[i], yerr=liste_ectypejour[i] / np.sqrt(5 * nb_participant), xerr=0,linewidth=2, capsize=6, c='black')
elif type.upper()=='VIBREUR' or type.upper()=='MOTEUR' or type.upper()=='VIBRATION' or type.upper()=='MANETTE':
    for i in range(nb_jour):
        moy_jouri = np.mean(liste_donnees[1][5*i*nb_participant:(5*i+5)*nb_participant])
        ec_jouri = np.std(liste_donnees[1][5*i*nb_participant:(5*i+5)*nb_participant])
        analyse_indiv_rapid(prenom)
        liste_moyennejour.append(moy_jouri)  # commence au jour 0 (1er jour de test)
        tdrevol_jour.append(liste_moyennejour[i] - liste_moyennejour[i - 1])
        liste_ectypejour.append(ec_jouri)
        eccevol_jour.append(liste_ectypejour[i] - np.mean(liste_ectypejour))
        axs[0].errorbar(i+1, liste_moyennejour[i], yerr=liste_ectypejour[i] / np.sqrt(5 * nb_participant), xerr=0, linewidth=2,capsize=6,c='black')
        axs[1].errorbar(i + 1, tdrevol_jour[i], yerr=liste_ectypejour[i] / np.sqrt(5 * nb_participant), xerr=0, linewidth=2, capsize=6, c='black')
if prenom.upper()=='TOUT':
    prenom="l'ensemble des participants"
```

Annexe II – 8/13 Programme python traitement des données

```
axs[0].grid()
axs[0].set_ylim((0, 350))
axs[0].set_xticks(np.arange(0, nb_jour + 2, 1))
axs[0].set_xlabel("Jour d'étude",fontsize=17)
axs[0].set_ylabel('Temps de réaction (ms)',fontsize=17)
axs[0].plot(liste_jour, liste_moyennejour,marker='o', linestyle='-', c='red', label='Temps de réaction moyens de ' + prenom)
axs[0].plot(liste_jour, liste_ectypejour, marker='o', linestyle='-', c='b',label="Écart type moyens de " + prenom)
axs[0].legend(fontsize=15)
axs[0].set_title("Évolution de la moyenne du temps de réaction par jour", fontsize=22)

axs[1].grid()
axs[1].set_ylim((-80, 80))
axs[1].set_xticks(np.arange(0, nb_jour + 2, 1))
axs[1].set_xlabel("Jour d'étude",fontsize=17)
axs[1].set_ylabel('Delta du temps de réaction (ms)',fontsize=17)
axs[1].plot(liste_jour, tdrevol_jour, marker='o',linestyle='-', c='orange', label='Delta du temps de réaction de ' + prenom)
axs[1].plot(liste_jour, eccevol_jour, marker='o', linestyle='-', c='b',label="Delta de l'écart type de " + prenom)
axs[1].legend(fontsize=15)
axs[1].set_title("Évolution du delta de temps de réaction par jour", fontsize=22)

moy_tdrevol_jour = np.mean(tdrevol_jour[1:])
plt.tight_layout()
plt.show()
print("La moyenne de l'évolution du tdr par jour de "+prenom+" est de " +str(round(moy_tdrevol_jour,1))+' ms.')
print("L'evolution du temps de réaction depuis le début est de " + str(round(liste_moyennejour[0]-liste_moyennejour[-1],1))+ ' ms.')
```

Annexe II – 9/13 Programme python traitement des données

```python
def inverse_moyenne_tdr_intens(data):
    '''
    Créer un dictionnaire de l'inverse de la moyenne des résultats aux tests pour chaque rapport cyclique
    :param data: Dictionnaire des résultats en fonctions des rapports cycliques
    :return:
    '''
    return {key: 1 / np.mean(values) for key, values in data.items()}
```

2 usages

```python
def normaliser(x):
    '''
    Permet de normer les données pour ne pas avoir des valeurs trop élevées lors de la descente de gradient
    '''
    return (x - np.mean(x)) / np.std(x)

def cout(theta, x_biais, y):
    '''
    Fonction cout de la descente de gradient
    :param theta: Paramètres liant x_b et y (Ils évoluent durant la descente de gradient)
    :param x_biais: Colonne des entrées avec une colonne de 1 pour prendre les biais (la constante dans l'équation de la droite)
    :param y: Sortie
    :return:
    '''
    m = len(y)
    return (1 / (2 * m)) * np.sum((x_biais.dot(theta) - y.reshape(-1, 1)) ** 2)
```

Annexe II – 10/13 Programme python traitement des données

```python
def descente_gradient(rapport_cyclique, tdr, alpha=0.01, n_iterations=1000):
    '''
    Fonction effectuant la descente de gradient sur un test de données
    :param rapport_cyclique: Entrées
    :param tdr: Sortie
    :param alpha: Taux d'apprentissage
    :param n_iterations: Nombres d'itérations de la descente de gradient
    :return:
    '''
    rapport_cyclique_norm = normaliser(rapport_cyclique)
    tdr_norm = normaliser(tdr)
    nb_points = len(rapport_cyclique_norm)
    theta = np.random.randn(2, 1)
    rapport_cyclique_b = np.c_[np.ones((nb_points, 1)), rapport_cyclique_norm.reshape(-1, 1)]
    for iteration in range(n_iterations):
        gradients = (2 / nb_points) * rapport_cyclique_b.T.dot(rapport_cyclique_b.dot(theta) - tdr_norm.reshape(-1, 1))
        theta = theta - alpha * gradients
    return theta, rapport_cyclique_b, np.mean(tdr), np.std(tdr)
```

Annexe II – 11/13 Programme python traitement des données

```
def afficher_regression(noms_selectionnes=None,comparaison=False):
    '''
    Permet d'afficher sur un graphique les regressions en 1/x du tests de données. Il peut aussi comparer sur un même graphique différentes régressions.
    Et peut afficher la régression de l'ensemble des données
    :param noms_selectionnes: Liste des noms qui doivent être pris en compte pour la/les régressions (Si on les compare ou on les regroupe)
    :param comparaison: False --> Pas de comparaison, donc si plusieurs nom, on a une régression sur l'ensemble des valeurs de noms_selectionnes.
    True --> Comparaison entre les régressions de noms_selectionnes.
    :return:
    '''
    fig, axs = plt.subplots( nrows: 1,  ncols: 2, figsize=(18, 9))
    ec_intens=[]
    if noms_selectionnes is None:
        noms_selectionnes = list(res_gradient_vib.keys())

    combined_tdr = []
    combined_rapport_cyclique = []
    ec_i=[]
    points = np.linspace( start: 1,  stop: 1500,  num: 1000)
    for nom in noms_selectionnes:
        moy_tdr = inverse_moyenne_tdr_intens(res_gradient_vib[nom])
        rapport_cyclique = np.array(list(moy_tdr.keys()))
        tdr = np.array(list(moy_tdr.values()))
        for keys in list(res_gradient_vib[nom].keys()):
            moy_ec_intens[keys].extend(res_gradient_vib[nom][keys])
            ec_i=np.std(res_gradient_vib[nom][keys])
        # Tracer les points de données d'origine (TDR inversé)
        line, =axs[0].plot(rapport_cyclique, 1/tdr, ".", label=f"{nom.capitalize()} data")
        couleur_point=line.get_color()
        axs[0].errorbar(rapport_cyclique, 1/tdr, yerr = ec_i / np.sqrt(len(res_gradient_vib[nom][440])), xerr = 0, fmt = 'none', linewidth = 1, capsize = 3,c=couleur_point)
        # Accumuler les données pour la combinaison
        combined_tdr.extend(tdr)
        combined_rapport_cyclique.extend(rapport_cyclique)
```

Annexe II – 12/13 Programme python traitement des données

```
    if comparaison:
        # Tracer la courbe de régression pour un seul participant
        theta, rapport_cyclique_b, mean_tdr, std_tdr = descente_gradient(rapport_cyclique, tdr)
        # Expression de la fonction tracée
        m_i = theta[1][0] * std_tdr / np.std(rapport_cyclique)  # Coefficient de pente
        b_i = (theta[0][0] * std_tdr + mean_tdr) - m_i * np.mean(rapport_cyclique)  # Ordonnée à l'origine
        axs[0].plot(points, 1 / (m_i * points + b_i), label=f"{nom.capitalize()}")
for keys in moy_ec_intens.keys():
    ec_intens.append(np.std(moy_ec_intens[keys]))

if len(noms_selectionnes) == 1:
    # Tracer la courbe de régression pour un seul participant
    theta, rapport_cyclique_b, mean_tdr, std_tdr = descente_gradient(rapport_cyclique, tdr)
    # Expression de la fonction tracée
    m = theta[1][0] * std_tdr / np.std(rapport_cyclique)  # Coefficient de pente
    b = (theta[0][0] * std_tdr + mean_tdr) - m * np.mean(rapport_cyclique)  # Ordonnée à l'origine
elif len(noms_selectionnes) > 1:
    # Tracer la courbe de régression combinée pour plusieurs participants
    combined_tdr = np.array(combined_tdr)
    combined_rapport_cyclique = np.array(combined_rapport_cyclique)
    theta_combined, rapport_cyclique_b_combined, mean_combined_tdr, std_combined_tdr = descente_gradient(
        combined_rapport_cyclique, combined_tdr)
    # Expression de la fonction tracée
    m = theta_combined[1][0] * std_combined_tdr / np.std(combined_rapport_cyclique)  # Coefficient de pente
    b = (theta_combined[0][0] * std_combined_tdr + mean_combined_tdr) - m * np.mean(
        combined_rapport_cyclique)  # Ordonnée à l'origine
```

Annexe II – 13/13 Programme python traitement des données

```
axs[0].plot(points,1/(m*points+b),c='b',label='Régression linéaire par descente de gradient')
axs[0].plot(points,100*points/points,c='r',label='Limite théorique du temps de réaction')
axs[0].set_xlabel("Rapport cyclique (nb binaire 0-1023)",fontsize=17)
axs[0].set_ylabel("Temps de réaction (ms)",fontsize=17)
axs[0].set_ylim((100,701))
axs[0].set_xlim(200,1300)
axs[0].set_xticks(np.arange(200,1300,80))
axs[0].set_yticks(np.arange(40,700,60))
axs[0].legend(fontsize=14)
axs[0].grid()
axs[0].set_title(f"Temps de réaction en fonction du rapport cyclique",fontsize=22)

axs[1].plot(rapport_cyclique, ec_intens, c='b',label='Écarts types moyens')
axs[1].set_xlabel("Rapport cyclique (nb binaire 0-1023)",fontsize=17)
axs[1].set_ylabel("Écarts types moyens (ms)",fontsize=17)
axs[1].set_ylim((0, 120))
axs[1].set_xlim(400, 1040)
axs[1].set_xticks(np.arange(400, 1040, 40))
axs[1].set_yticks(np.arange(0, 120, 10))
axs[1].legend(fontsize=18)
axs[1].grid()
axs[1].set_title(f"Écarts types moyens en fonction du rapport cyclique",fontsize=22)
plt.tight_layout()
plt.show()
print("L'équation de la courbe est " + f"y =1/({m:.2e}x + {b:.2e})")
```

Annexe III – 1/4 - Programme python ESP 32

```
from random import randint
from machine import UART,Pin,PWM
import time
switch=Pin(19,mode=Pin.IN,pull=Pin.PULL_UP)
pwm_buzz=PWM(Pin(18),freq=2000,duty=0)
pwm_moteur = PWM(Pin(23), freq=2000, duty=0)
IN1=Pin(22,mode=Pin.OUT)
IN2=Pin(21,mode=Pin.OUT)
IN1.value(0)
IN2.value(0)
tr=[]
deltatr=[]
liste=[]  # initialisation liste des  temps  des sollicitations
start=0
def sens1(f):
    pwm_moteur.duty(f)
    IN1.value(0)
    IN2.value(1)
def stop():
    IN1.value(0)
    IN2.value(0)
def duree_sollicitation(l): # renvoie la duree maximale de sollicitation
    duree_max=l[-1]                 # sur la liste l
    for i in range(1,len(l)-1):
        if l[i+1]-l[i]<duree_max:
            duree_max=l[i+1]-l[i]
    return duree_max
```

Annexe III – 2/4 - Programme python ESP 32

```
def renvoi_temps(dt,n,d,a):# renvoie la liste des temps de sollicitation
    pas=int(dt*1000/(n+1))
    l=list(range(pas,(dt)*999,pas))
    if a==0: # aleatoire
        flag=True
        while flag:
            for i in range(len(l)):
                k=randint( a: 0, b: 100)
                l[i]+=int(pas/100)*k
            if duree_sollicitation(l)>d:
                flag=False
    else:
        pass
    return l
def mesure_temps(switch):# envoi vers le pc a chaque appui de switch
    global tr,liste,start
    chaine=''
    t=time.ticks_diff(time.ticks_ms(), start)
    if len(tr)!=0:
        if (t-tr[-1])>200: #pour eviter le rebond du switch
            tr.append(t)
            t_liste=liste[len(tr)-1]
            delta=t-t_liste
            chaine=str(delta)+','+str(t_liste)
            print(chaine)
            uart.write(chaine.encode())
    else:
        tr.append(t)
        delta=t-liste[0]
        chaine=str(delta)+','+str(liste[0])
        print(chaine)
        uart.write(chaine.encode())
```

Annexe III – 3/4 - Programme python ESP 32

```
# duree observation : dt  en s
# nbre d'impulsions :n
# duree de sollicitation: d ( en ms)
# Buzzer Manette: s  (s=0 : buzzer  s=1 : manette )
# variable aleatoire: a  ( a=0 aleatoire)
# intensite de la sollicitation: f ( 0<i<1024  i entier)
# chaine_envoi=dt,n,d,s,a,f
# dt,n,d,s,a,f=10,4,200,True,True,800

def sollicitation(dt,n,d,s,a,f):
    global tr,liste,start
    liste=renvoi_temps(dt,n,d,a)
    tr=[]
    start = time.ticks_ms()
    for i in liste:
        while  time.ticks_diff(time.ticks_ms(), start)<=i:
            pass
        startbis=time.ticks_ms()
        if s==0:
            pwm_buzz.duty(f)
            while  time.ticks_diff(time.ticks_ms(), startbis)<=d:
                pass
            pwm_buzz.duty(0)
        if s==1:
            sens1(f)
            while  time.ticks_diff(time.ticks_ms(), startbis)<=d:
                pass
            stop()
switch.irq(handler=mesure_temps,trigger=Pin.IRQ_FALLING)
command="liaison ok "
retour=''
attente=True
```

Annexe III – 4/4 - Programme python ESP 32

```
while attente:
    try:
        uart = UART(2,115200 )
        time.sleep_ms(300)
        retour =uart.read().decode('utf8')
        time.sleep_ms(100)
        if retour!='':
            print(retour)
            ch=retour.strip().split(',')
            dt,n,d,s,a,f=int(ch[1]),int(ch[2]),int(ch[3]),int(ch[4]),int(ch[5]),int(ch[6])
            print("dt,n,d,s,a,f :  ",dt,n,d,s,a,f)
            uart.write(command.encode())
            attente=False
    except:
        pass
print("connexion ok")
while True:
    try:
        data=uart.read().decode('utf8')
        print(data)
        if data[:6]=="config":
            ch=data.strip().split(',')
            dt,n,d,s,a,f=int(ch[1]),int(ch[2]),int(ch[3]),int(ch[4]),int(ch[5]),int(ch[6])
            print("dt,n,d,s,a,f :  ",dt,n,d,s,a,f)
        if data=='on':
            sollicitation(dt,n,d,s,a,f)
        if data=='off':
            break
    except:
        pass
```

Annexe IV - Fabrication par impression 3D

Figure IV.1 – Manette – échecs impression 3D

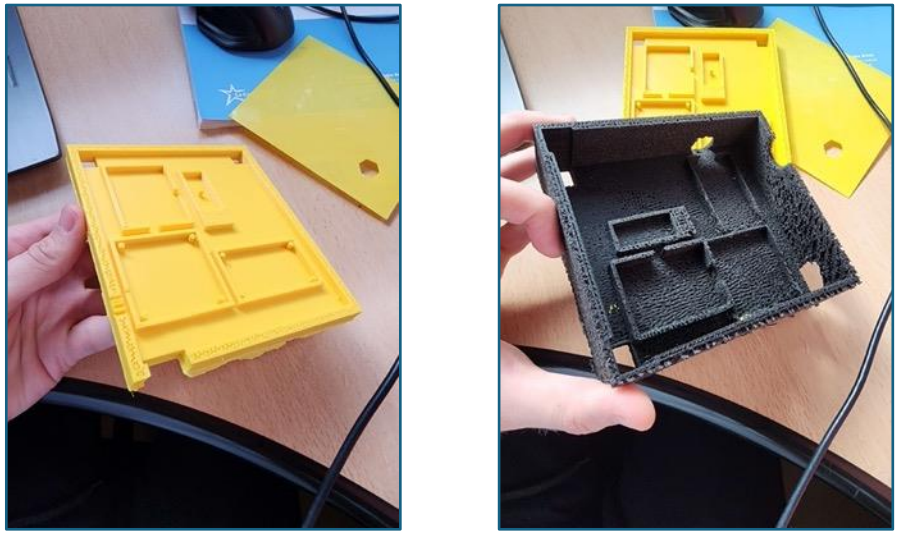


Figure IV.2 – Boîte unité centrale – tentatives d'impression 3D

Annexe V - Imprimante ALM (additive layer manufacturing)

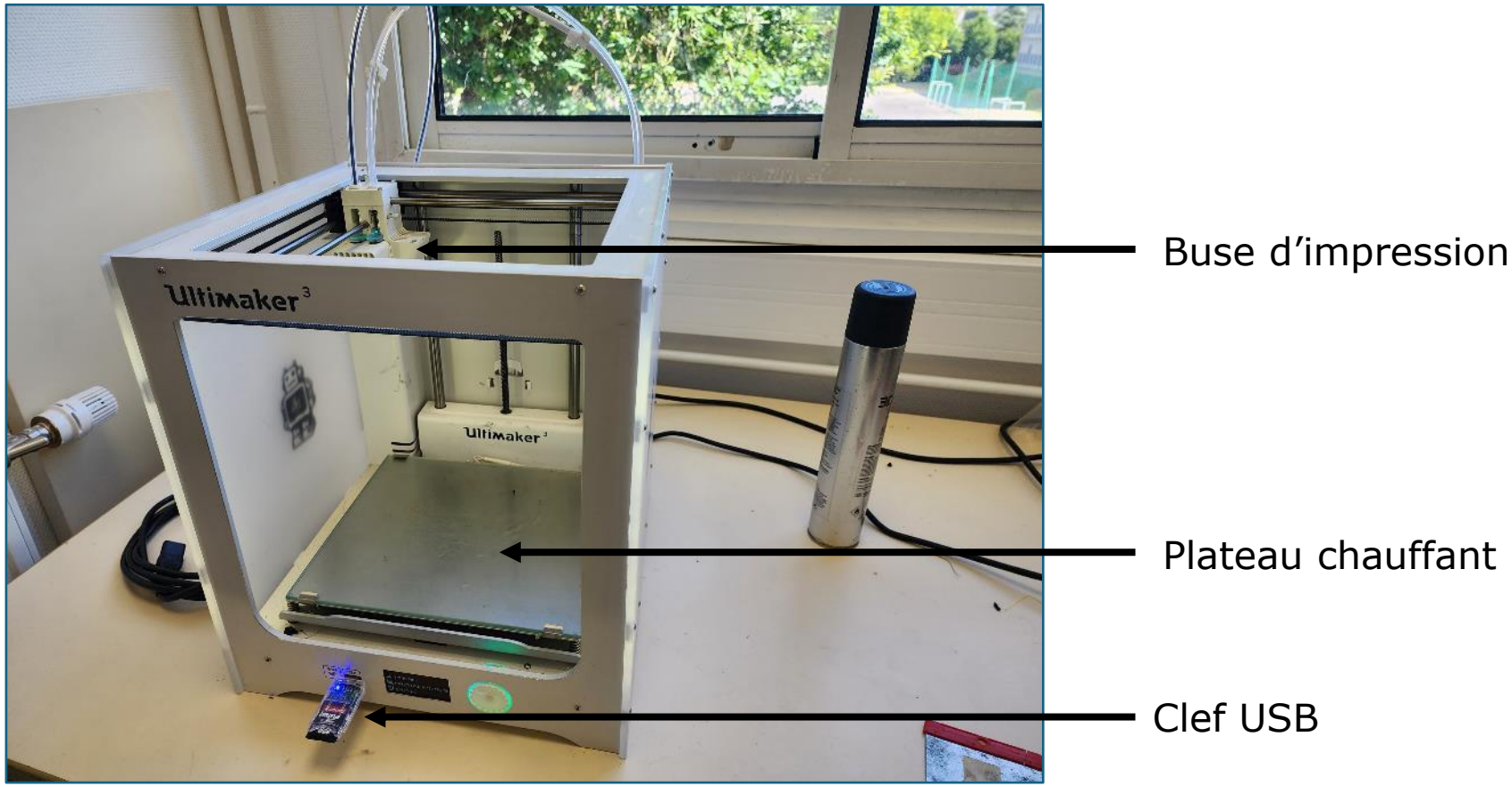


Figure V.1 – Imprimante 3D Ultimaker 3

Annexe VI - Résultats de Henri Piéron de 1932

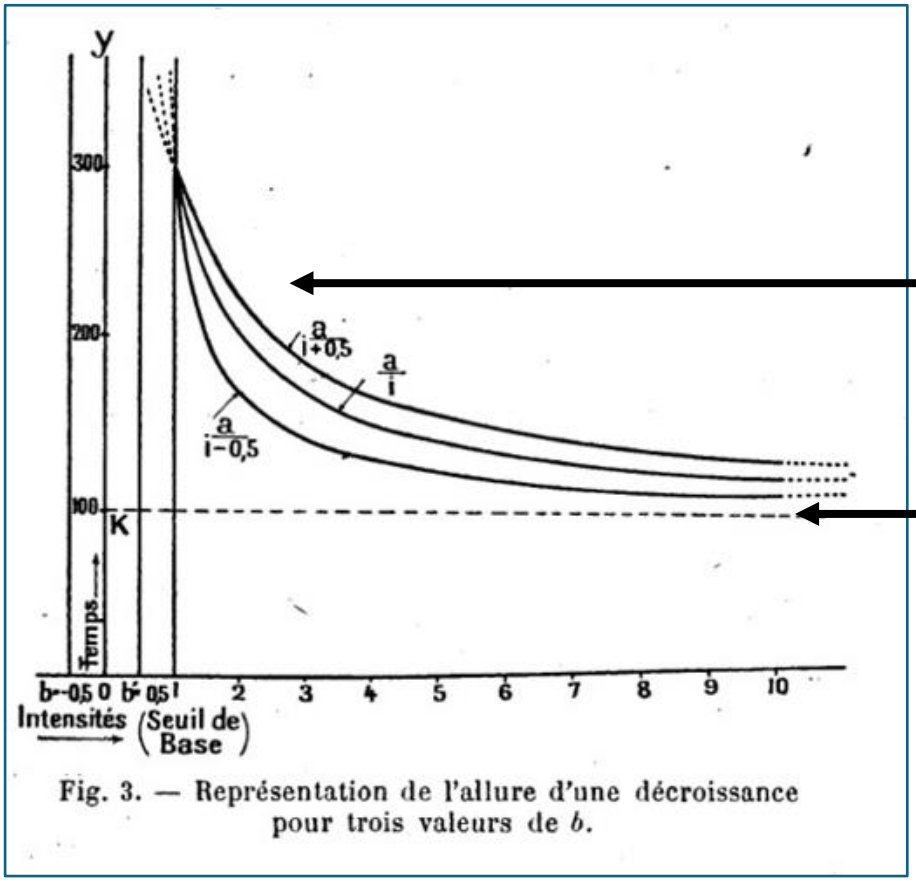


Fig. 3. — Représentation de l'allure d'une décroissance pour trois valeurs de *b*.

Courbe en $\frac{1}{x}$

Limite théorique humaine (100ms)

Figure VI.1 – Représentation de l'allure d'une décroissance (H.Piéron)