Système d'Optimisation des Menus des Sportifs

ANNEXES

Nom et Prénom	N° SCEI	Session
SOUADI Nouamane	18516	2024

Liste des figures

A1 : code d'identification

```
// Déclaration des broches du codeur incrémental
const int pinA = 2; // Broche A du codeur connectée à la broche 2 d'Arduino (interrupt 0)
const int pinB = 3; // Broche B du codeur connectée à la broche 3 d'Arduino
const int IN2 = 10;
const int IN1 = 11;

// Variables pour le comptage des impulsions du codeur
int pulseCount = 0; // Compteur d'impulsions
int previousPulseCount = 0; // Nombre d'impulsions précédent pour le calcul de la vitesse
unsigned long previousMillis = 0; // Temps précédent pour le calcul du délai
unsigned long intervalle = 50000; // Intervalle de temps pour le calcul de la vitesse (en Microsecondes)
int pulseParRevolution = 48;

void setup() {
  // Initialisation des broches du codeur
  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  // Activation de l'interruption pour la broche A (interrupt 0)
  attachInterrupt(digitalPinToInterrupt(pinA), countPulse, RISING);
  attachInterrupt(digitalPinToInterrupt(pinB), countPulse, RISING);
  attachInterrupt(digitalPinToInterrupt(pinA), countPulse, FALLING);
  attachInterrupt(digitalPinToInterrupt(pinB), countPulse, FALLING);

  // Démarrage de la communication série
  Serial.begin(9600);
  digitalWrite(IN1, 1);
  digitalWrite(IN2, 0);
  Serial.println("t;w(t)");
  Serial.println("0;0");
}
void loop() {
  unsigned long currentMillis = micros();

  // Calcul du temps écoulé depuis le dernier calcul de vitesse
  if (currentMillis - previousMillis >= intervalle) {
    // Enregistrement du nombre d'impulsions pendant l'intervalle
    float A = pulseCount - previousPulseCount;
    // Calcul de la vitesse en impulsions par seconde
    float speed = 2*PI*A / intervalle / pulseParRevolution * 1000000.0; // rad/s

    // Affichage de la vitesse
    Serial.print(currentMillis);
    Serial.print(";");
    Serial.println(speed);

    // Mise à jour des variables pour le prochain calcul de vitesse
    previousMillis = currentMillis;
    previousPulseCount = pulseCount;
  }
}

void countPulse() {
  // Augmentation du compteur d'impulsions à chaque interruption
  pulseCount++;
}
```

A2 : régression linéaire de K et C_m

regression U(w).py > ...

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv("ressources/liste U(w).txt", delimiter=";")
x, y = np.array(data['w']), np.array(data["U"])
[a, b] = np.polyfit(x, y, 1)
def fct(x):
    return a*x + b
R = 9.3 # Par multimètre
Cm = b*a/R
print(f"a = K = {a} V.s/rad, Cm = b.K/R = {Cm} N.m")

x2 = np.linspace(min(x), max(x), 2)

plt.plot(x, y, 'o', label="acquisition U(w)")
plt.plot(x2, fct(x2), label="polyfit")
plt.title(f"$K$=a={round(a, 2)} V.s/rad, $C_m$=b.K/R={round(Cm, 3)} N.m")
plt.legend()
plt.grid()
plt.show()
```

A3 : code de régression de L

regression L.py > ...

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

"""
Forme i(t) = [1-exp(-t/taux)]Iper avec Iper = U/Re
Re = Rm + 220
"""
Re = 9.3 + 220
data = pd.read_csv("ressources/acquisitionL.txt", delimiter=",")
t, u = np.array(data["t"]), np.array(data["Ur(t)"])
i = u/Re

indice0 = 0
while t[indice0] < 0:
    indice0 += 1
t = t[indice0:]
i = i[indice0:]

def f(t, Iper, taux):
    return Iper*(1-np.exp(-t/taux)) # taux = L/Re
[Iper, taux], val = curve_fit(f, t, i)
plt.plot(t*1e3, i*1e3, label="acquisition")
plt.plot(t*1e3, f(t, Iper, taux)*1e3, label="curve_fit")
plt.xlabel("t (ms)")
plt.ylabel("i(t) (mA)")
L = Re*taux
plt.title(f"L = Re/taux = {round(L, 3)}H")
print(f"L= {L} H")
plt.show()
```

élimination de la partie négative du temps

A4 : code d'asservissement (arduino)

```
// Déclaration des broches du codeur incrémental
const int pinA = 2; // Broche A du codeur connectée à la broche 2 d'Arduino (interrupt 0)
const int pinB = 3; // Broche B du codeur connectée à la broche 3 d'Arduino
const int IN2 = 10;
const int IN1 = 11;
const int ENA = 9;

// Variables pour le comptage des impulsions du codeur
volatile int pulseCount = 0; // Compteur d'impulsions
int previousPulseCount = 0; // Nombre d'impulsions précédent pour le calcul de la vitesse
unsigned long previousMillis = 0; // Temps précédent pour le calcul du délai
unsigned long interval = 50000; // Intervalle de temps pour le calcul de la vitesse (en Microsecondes)
int pulseParRevolution = 48;

// Variable asservissement
const float kp = 4310;
const float ki = 4310/1.32;
const int max_pwm = 255;
const int teta_c = 180;
float teta = 0;
float erreur = 0;
float somme_erreur = 0;
float speed = 0;
float var = 0;
const float r = 0.057438;

void setup() {
  // Initialisation des broches du codeur
  pinMode(pinA, INPUT);
  pinMode(pinB, INPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
  // Activation de l'interruption pour la broche A (interrupt 0)
  attachInterrupt(digitalPinToInterrupt(pinA), countPulse, RISING);
  attachInterrupt(digitalPinToInterrupt(pinB), countPulse, RISING);
  attachInterrupt(digitalPinToInterrupt(pinA), countPulse, FALLING);
  attachInterrupt(digitalPinToInterrupt(pinB), countPulse, FALLING);

  // Démarrage de la communication série
  Serial.begin(9600);
  Serial.println("t;teta(t);wr(t);erreur;pwm");
  Serial.print("0;0.00;0.00;");
  Serial.print(teta_c-teta);
  Serial.println(";0.00");
}

void loop() {
  unsigned long currentMillis = micros();

  // Calcul du temps écoulé depuis le dernier calcul de vitesse
  if (currentMillis - previousMillis >= interval) {
    // bloc d'asservissement
    teta += speed * interval / 1000000.00;
    erreur = teta_c - teta;
    somme_erreur += erreur * interval / 1000000.00;
    var = kp * erreur + ki * somme_erreur;

    // Enregistrement du nombre d'impulsions pendant l'intervalle
    float A = pulseCount - previousPulseCount;
    // Calcul de la vitesse en impulsions par seconde
    speed = 360*(float)A / (float)interval / pulseParRevolution * 1000000.0; // °/s
```

```
    speed *= r; // sortie reducteur

    if (var > 0) {
      digitalWrite(IN1, 1);
      digitalWrite(IN2, 0);
    } else {
      digitalWrite(IN1, 0);
      digitalWrite(IN2, 1);
      speed *= -1;
      var *= -1;
    }

    if (var > max_pwm){
      var = max_pwm;
    }
    analogWrite(ENA, var);

    // Affichage
    Serial.print(currentMillis);
    Serial.print(";");
    Serial.print(teta);
    Serial.print(";");
    Serial.print(speed);
    Serial.print(";");
    Serial.print(teta_c-teta);
    Serial.print(";");
    Serial.println(var);

    // Mise à jour des variables pour le prochain calcul de vitesse
    previousMillis = currentMillis;
    previousPulseCount = pulseCount;
  }
}

void countPulse() {
  // Augmentation du compteur d'impulsions à chaque interruption
  pulseCount++;
}
```

A5: code RFID (arduino)

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9

MFRC522 rfid(SS_PIN, RST_PIN); // Instance of the class
MFRC522::MIFARE_Key key;
byte nuidPICC[4];
void setup() {
  Serial.begin(9600);
  SPI.begin(); // Init SPI bus
  rfid.PCD_Init(); // Init MFRC522
}

void loop() {

  // Reset the loop if no new card present on the sensor/reader. This saves the entire process when idle.
  if ( ! rfid.PICC_IsNewCardPresent())
    return;

  // Verify if the NUID has been readed
  if ( ! rfid.PICC_ReadCardSerial())
    return;
    printHex(rfid.uid.uidByte, rfid.uid.size);
    Serial.println();
  // Halt PICC
  rfid.PICC_HaltA();
  // Stop encryption on PCD
  rfid.PCD_StopCrypto1();
```

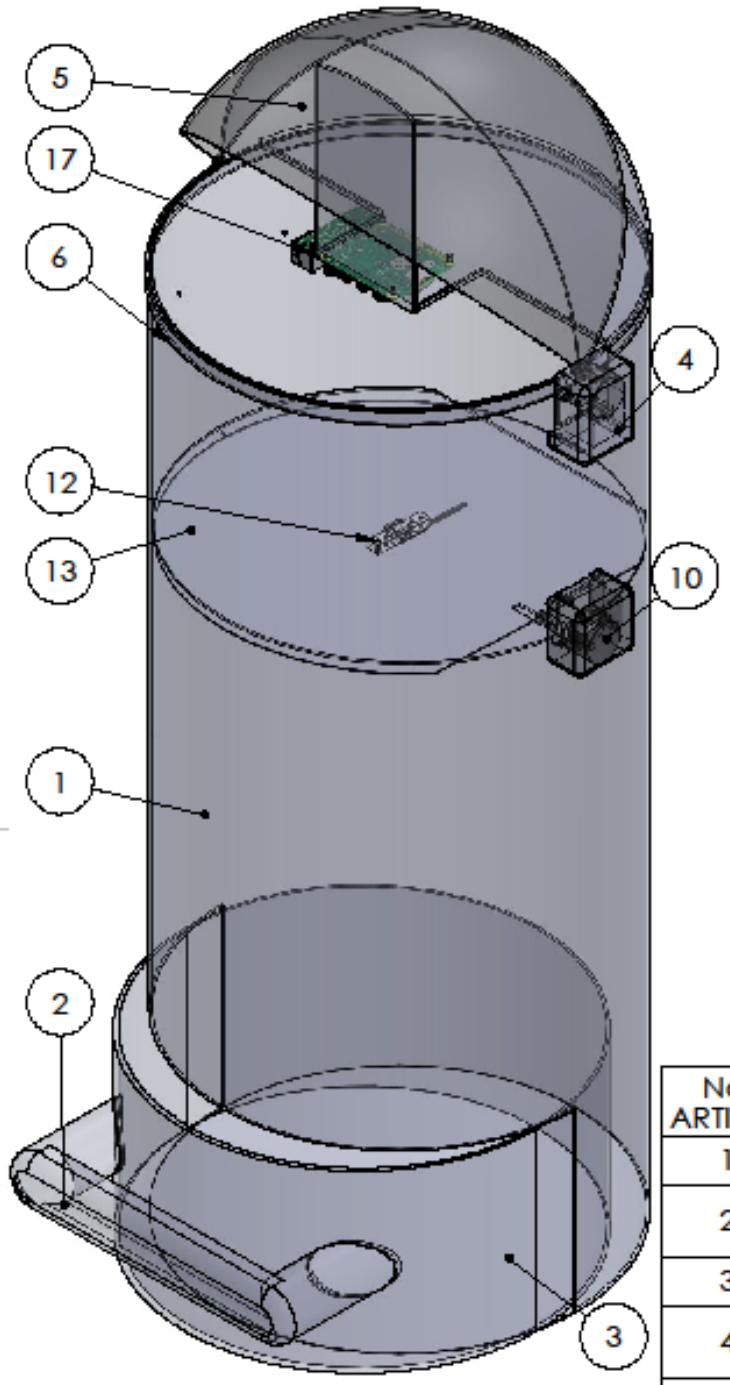
```
}
void printHex(byte *buffer, byte bufferSize) {
  for (byte i = 0; i < bufferSize; i++) {
    Serial.print(buffer[i] < 0x10 ? " 0" : " ");
    Serial.print(buffer[i], HEX);
  }
}
```

A6: code RFID (python)

GUI TIPE.py > afficher_adherent_et_repas

```python
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk
import sqlite3
import serial

# Create the main window
fenetre = tk.Tk()

bg = "#99D9EA"
fenetre.configure(bg=bg)

# Calculate the coordinates to center the window on the screen
largeur_ecran = fenetre.winfo_screenwidth()
hauteur_ecran = fenetre.winfo_screenheight()
x = (largeur_ecran - 700) // 2
y = (hauteur_ecran - 700) // 2

# Position the window at the center of the screen
fenetre.geometry(f"700x700+{x}+{y}")

def afficher_adherent_et_repas(id, fenetre):
    try:
        # Connect to the database
        conn = sqlite3.connect('sqlite_TIPE.db')
        conn.row_factory = sqlite3.Row
        cur = conn.cursor()
    except sqlite3.Error as e:
        print(f"Erreur de connexion à la base de données: {e}")
        return

    cur.execute("SELECT * FROM adherent WHERE id=?", (id,))
    adherent = cur.fetchone()

    if adherent is None:
        print(f"Aucun adhérent trouvé avec cet ID: '{id}'")
        conn.close()
        return

    print(f"ID détecté: '{id}'")

    # Clear the previous contents of the window
    for widget in fenetre.winfo_children():
        widget.destroy()

    # Create a frame to group the adherent and repas information
    frame_principal = tk.Frame(fenetre, bg=bg)
    frame_principal.pack(expand=True, fill="both")

    # Add a title label
    label_titre = tk.Label(frame_principal, text="Informations de l'adhérent et du repas", font=("Helvetica", 16),
                           bg=bg)
    label_titre.pack(pady=10)

    # Add the adherent information
    # Image de profil en haut à droite
    img_profil = Image.open(adherent['img_profil'])
```

```
    img_profil = img_profil.resize((100, 100))
    img_profil_tk = ImageTk.PhotoImage(img_profil)
    label_img_profil = tk.Label(frame_principal, image=img_profil_tk, bg=bg)
    label_img_profil.pack(side="right", anchor="n", padx=10, pady=10)

    # Nom et prénom à gauche de l'image de profil
    label_nom_prenom = tk.Label(frame_principal, text=f"Nom: {adherent['nom']}\nPrénom: {adherent['prenom']}",
                                font=("Helvetica", 14), bg=bg)
    label_nom_prenom.pack(side="left", anchor="n", padx=10, pady=10)

    # Add the repas information
    # Table des items avec leurs quantités (centrée)
    cur.execute("SELECT item, quantite FROM plat WHERE plat_id=?", (id,))
    items_quantites = cur.fetchall()

    # Créer une Treeview (table) avec des colonnes "Élément" et "Quantité"
    table_items = ttk.Treeview(frame_principal, columns=("Element", "Quantite"), show="headings")
    table_items.heading("Element", text="Élément")
    table_items.heading("Quantite", text="Quantité")

    for item_quantite in items_quantites:
        table_items.insert("", "end", values=(item_quantite['item'], item_quantite['quantite']))

    table_items.pack(side="bottom", anchor="center", pady=20)

    # Image du plat centrée
    cur.execute("SELECT plat_id, plat_img FROM plat WHERE plat_id=?", (id,))
    plat = cur.fetchone()

    if plat:
        img_plat = Image.open(plat['plat_img'])
        img_plat = img_plat.resize((200, 200))
        img_plat_tk = ImageTk.PhotoImage(img_plat)
        label_img_plat = tk.Label(frame_principal, image=img_plat_tk, bg=bg)
        label_img_plat.pack(side="bottom", pady=20)

    conn.close()
    fenetre.update()

def read_rfid_data():
    while True:
        try:
            rfid_data = arduino_serial_port.readline().decode('utf-8').strip()
            afficher_adherent_et_repas(rfid_data, fenetre)
        except serial.SerialException as e:
            print(f"Erreur lecture du port série: {e}")


# Open the serial port
arduino_serial_port = serial.Serial('COM3')

# Start reading RFID data
read_rfid_data()

# Run the main loop
fenetre.mainloop()
```



No. ARTICLE	NUMERO DE PIECE	QTE
1	corps	1
2	corps (poignet poubelle)	1
3	corps (poubelle)	1
4	support moteur tête	1
5	tête (mobile)	1
6	tête (fixe)	1
10	support déchets (moteur)	1
12	MICRO_LOAD_CELL_FMHD-1kg	1
13	support capteur déchets	1
17	PCB, RPi4ModelB	1

ECHELLE 1:4