```
# -*- coding: utf-8 -*-
##############################################################################################
##############################################################################################
##############################################################################################
##############################################################################################

# Importamos librerias necesarias para el programa
import serial
import subprocess
from time import sleep
import os
import pymysql

# Variables globales
leer = True
comando= ""          # Iremos almacenando el comando que leeamos
connected=False      # Flag de detección de mega conectada
start_char = "$"     # Caracter que indica el inicio del comando
end_char = "#"       # Caracter que indica final del comando
char_lim = '*'       # Caracter delimitador de comandos

# Lista con los errores de la calibracion
errores=[]
# Lista con la posicion de los home nuevos
homes=[]
# numero de serie del arduino
num_serie = ""

# Variables relacionadas con las BBDD
# Datos del servidor
config = {
  'user': 'root',
  'passwd': 'toor',
  'host': '172.16.16.15',
  'db': 'zowi',
}
# Datos del localhost
config_localhost = {
  'user': 'root',
  'passwd': 'toor',
  'host': '127.0.0.1',
  'db': 'zowi',
```

```
}

# Tabla usada
table = 'calibracion'
# Secuencia SQL
sql = "INSERT INTO " + table + " (serial_number,e_left_hip, e_right_hip, e_left_foot, e_right_foot,
       state, h_left_hip, h_right_hip, h_left_foot, h_right_foot) VALUES (%s, %s, %s, %s, %s,%s,
       %s, %s, %s,%s)"

def funcionPuertos():
    status = 0
    try:
       subprocess.check_call("echo '1-1.5' | sudo tee /sys/bus/usb/drivers/usb/unbind", shell=True)
    except:
        status+=1
       print("Error unbind")
    else:
       print("OK unbind")
    #sleep(1)
    try:
       subprocess.check_call("echo '1-1.5' | sudo tee /sys/bus/usb/drivers/usb/bind", shell=True)
    except:
        status+=2
       print("Error bind")
    else:
       print("OK bind")
    #sleep(1)
    return status
##########################################################################################
##########################################################################################
##########################################################################################

    # FUNCIoN MAIN()
try:
    ##########################################################################################
    # Nos conectamos a la base de datos remota
    #print("Nos conectamos a la BBDD")
    #try:
    #    pass
       # Quitado la conexion a la base de datos remota
       #db = pymysql.connect(**config)
       # Si hay algun error en la conexion devolvemos un error
```

```
#except pymysql.DatabaseError as e:
#    print("Error %d: %s" % (e.args[0], e.args[1]))
# Si no se produce ningun error continuamos la ejecucion
#else:
#    # Automaticamente hace un commit de los queries que reciba
#    db.autocommit(1)
#    print("Conectado a la BBDD remota")
#    cursor = db.cursor()

   ###
    # Escribimos en las bbdd remota un mensaje de inicio
#    try:
#         # mandamos directamente el diccionario donde están almacenados los errores.
#             # Los valores de errores, se mandan tal cual son recibidos de la Mega
#             cursor.execute( sql, (1, 1,1,1, 1,1,1, 1,1,1))
#             # Si hay algun error en la conexion devolvemos un error
#    except pymysql.DatabaseError as e:
#             print("Error %d: %s" % (e.args[0], e.args[1]))
#      # Si no hemos podido conectar a la BBDD al inicio:
#    except NameError:
#             print("No se pudo conectar al  base de datos remota en el inicio")
#    except:
#             print("Se ha producido un error al insertar la
#               sentencia en la base de datos remota")
#     # Si la sentencia se ha introducido correctamente
#    else:
#             print("Sentencia introducida correctamente en remoto")
#    #Cerramos el curso
#              #cursor.close()
   #####

# Codigo de inserccion de la base de datos local
try:
    db_localhost = pymysql.connect(**config_localhost)
# Si hay algun error en la conexion devolvemos un error
except pymysql.DatabaseError as e:
    print("Error %d: %s" % (e.args[0], e.args[1]))
# Si no se produce ningun error continuamos la ejecucion
else:
    # Automaticamente hace un commit de los queries que reciba
    db_localhost.autocommit(1)
    print("Conectado a la BBDD local")
```

```
    cursor_localhost = db_localhost.cursor()

   ###
   # Escribimos en la bbdd local mensaje de inicio
    try:
    #mandamos directamente el diccionario donde estan almacenados los errores.
    #Los valores de errores, se mandan tal cual son recibidos de la Mega
             cursor_localhost.execute( sql, (1, 1,1,1, 1,1,1, 1,1,1 ))
     #Si hay algun error en la conexion devolvemos un error
    except pymysql.DatabaseError as e:
             print("Error %d: %s" % (e.args[0], e.args[1]))
    # Si no hemos podido conectar a la BBDD al inicio:
    except NameError:
             print("No se pudo conectar al  base de datos local en el inicio")
    except:
             print("Se ha producido un error al insertar la
                       sentencia en la base de datos local")
    # Si la sentencia se ha introducido correctamente
    else:
             print("Sentencia introducida correctamente en local")
    #Cerramos el curso r
              #cursor_localhost.close()
     ########################################################
##############################################################################################
# Abrimos comunicacion con la placa de la MEGA
# en USB0 siempre va a ir la mega
print("Abriendo comunicacion con MEGA")
while(connected==False):
    try:
        mega = serial.Serial("/dev/mega",115200,timeout=0.2,dsrdtr=True )
    except serial.SerialException:
        print("Mega no conectada")
        sleep(5)
    else:
        # Limpiamos la información que haya en el serial

        # Espera obligatoria para reiniciar la mega
        sleep(5)
        print("Mandamos uno a la mega")
        mega.flushInput()
        mega.flushOutput()
        mega.write('1')
```

```
        sleep(0.5)
        connected=True
##############################################################################################
# LOOP()
##############################################################################################

print("Esperamos datos de MEGA")
while(1):
    data = mega.read()
    # La trama del mensaje  va entre  $ y # es decir:
    # $xxxxxxxxxxxxxxx#.
    # Por tanto leemos datos desde que recibimos un $ y leemos hasta el #
    # Estos caracteres se pueden cambiar ya que se almacenan en una variable
    while(data != start_char):
        data = mega.read()
    data = mega.read()
    while (data != end_char):
        comando = comando + data
        data = mega.read()
    print(comando)
    print("Mandamos datos a la zum")
    if (comando[:4] == "IZUM"):  # Comando de inicizalizacion de la zum
        comando = ""
        print("Vamos a programar la zum")
        # Programamos la ZUM
        status = funcionPuertos()
        if status == 1:
            try:
                cursor_localhost.execute(sql,(61,61,61,61,61,61,61,61,61,61))
            except:
                print("Error guardar 61")
        elif status == 2:
            try:
                cursor_localhost.execute(sql,(71,71,71,71,71,71,71,71,71,71))
            except:
                print("Error guardar 71")
        elif status == 3:
            try:
                cursor_localhost.execute(sql,(81,81,81,81,81,81,81,81,81,81))
            except:
                print("Error guardar 81")
        try:
```

```
            subprocess.check_call("avrdude -patmega328p -carduino -P/dev/zowi -b 115200 -D
                                   -Uflash:w:/home/pi/zowi/python/zowi_offset_i2c.cpp.hex:i",
                                   shell=True)
        # Si se produce algún error damos un mensaje de adventencia
        except subprocess.CalledProcessError:
         try:
             cursor_localhost.execute(sql,(4,4,4,4,4,4,4,4,4,4))
         except:
             print("Error guardar 4")
           print ("Programacion fallida")
           mega.write("M")
        # Si no se produce ningún error nos conectamos a la ZUM
        else:
            print("Conectando a zum")
            try:
                zum = serial.Serial("/dev/zowi",115200,timeout=0.2)
            except serial.SerialException:
                print("ZUM no conectada")
                mega.write("M")
            else:
                # Espera obligatoria para reiniciar la ZUM
                sleep(2)
                print("ZUM conectada")
                print("Esperando OK de ZUM")
                data = zum.read()
                while(data != start_char):
                    data = zum.read()
                data = zum.read()
                while (data != end_char):
                    comando = comando + data
                    data = zum.read()
                if (comando[:4]=="OKNS"):
                    num_serie = comando[5:]
                comando=""
                    mega.write("B")
                       print("B mandado")
                else:
                    comando = ""
                    mega.write("M")
                    print("M mandada")

    elif (comando[:4] == "MSxC"): # Comando con los datos de calibracion
```

```
    #No hacemos nada con estos comandos, simplemente los mandamos a la ZUM
     try:
             zum.write(comando)
     except NameError:
             print("conexion no establecida")
             #sleep(2)
     else:
             print("dato mandado")
elif (comando[:4] == "WERC"): #Comando con los datos de los errores en la calibracion
     #Parseamos la trama con los codigos, y los almacenamos en la lista de errores.
     #El orden de los errores es el siguiente:
     #error[0]=Cadera_izquierda
     #error[1]=Cadera_derecha
     #error[2]=Pie_izquierdo
     #error[3]=Pie derecho
     errores = comando[5:].split(char_lim)
elif (comando[:4] == "WOFC"): #Comando con datos de las posiciones para guardar en EEPROM
     print("COMANDO WOFC")
     #Tratamos la trama reibida para separar los valores de las posiciones home recibidas
     #Filtramos los 6 primeros caracteres que son el propio comando 'WOFC:*'
    homes=[]
     cadena = comando[6:]
     #Esperamos cuatro posiciones de home
     for i in range (4):
         #almacenamos las posiciones de home respecto de 90 en la lista home
         #El orden de los home es el siguiente:
         #homes[0]=Cadera_derecha
         #homes[1]=Pie_derecho
         #homes[2]=Cadera_izquierda
         #homes[3]=Pie izquierdo
      #Cambiado a nuevo 90-90
         # homes.append(90 - int( cadena[:3]))
         homes.append(int(cadena[:3])-90)
      #vamos borrando los datos tratados
         cadena = cadena[3:]
         #Una vez almacenadas las posiciones, mandamos el codigo a la zum
     try:
         zum.write(comando)
     except NameError:
         print("conexion no establecida")
         #Si se produce un error borramos las posiciones guardadas
         homes=[]
```

```
            #sleep(2)
        else:
            print("dato mandado")
        #añadir codigo para MYSQL
    elif (comando[:4] == "WSQL"):
        # Cerramos la comunicacion con la zum. La calibracion ha terminado
        zum.close()
        #Al recibir este comando escribimos en la base de datos
        try:
            #mandamos directamente el diccionario donde estan almacenados los errores.
            #Los valores de errores, se mandan tal cual son recibidos de la Mega
            cursor.execute( sql, (num_serie, errores[0], errores[1], errores[2], errores[3],
                                  errores[4], homes[2], homes[0], homes[3], homes[1]) )
        #Si hay algun error en la conexion devolvemos un error
        except pymysql.DatabaseError as e:
            print("Error %d: %s" % (e.args[0], e.args[1]))
            #Si la sentencia se ha introducido correctamente
        except NameError:
            print("No se pudo conectar a la base de datos remota en el inicio")
        except:
            print("Se ha producido un error al insertar la
                       sentencia en la base de datos remota")
        else:
            print("Sentencia introducida correctamente en remoto")
        #Al recibir este comando escribimos en la base de datos local para tener una copia
        try:
            #mandamos directamente el diccionario donde estan almacenados los errores.
            cursor_localhost.execute( sql, (num_serie, errores[0], errores[1], errores[2],
                                            errores[3], errores[4], homes[2],
                                            homes[0], homes[3], homes[1]) )
        #Si hay algun error en la conexion devolvemos un error
        except pymysql.DatabaseError as e:
            print("Error %d: %s" % (e.args[0], e.args[1]))
            #Si la sentencia se ha introducido correctamente
        except NameError:
            print("No se pudo conectar a la base de datos local en el inicio")
        except:
            print("Se ha producido un error al insertar la
                       sentencia en la base de datos local")
        else:
            print("Sentencia introducida correctamente en local")
            #Reseteamos las variables
```

```
            errores=[]
            homes=[]
            num_serie = ""
            #Cerramos el curso r
            #cursor_localhost.close()
            #Cerramos la conexión
            #db_localhost.close()
    elif (comando[:4] == "ROFF"):
        #COMANDO CON EL APAGADO SEGURO de la RASpBerry
        #subprocess.check_call("sudo halt",shell=True)
        print("Recibido comando de apagado controlado")

       ##################################################
        # Escribimos en las bases de datos un mensaje de fin
        try:
         # mandamos directamente el diccionario donde están almacenados los errores.
            # Los valores de errores, se mandan tal cual son recibidos de la Mega
            cursor.execute( sql, (0, 0,0,0, 0,0,0, 0,0,0))
            # Si hay algun error en la conexion devolvemos un error
        except pymysql.DatabaseError as e:
            print("Error %d: %s" % (e.args[0], e.args[1]))
        # Si no hemos podido conectar a la BBDD al inicio:
        except NameError:
            print("No se pudo conectar al  base de datos remota en el inicio")
        except:
            print("Se ha producido un error al insertar
                       la sentencia en la base de datos remota")
        # Si la sentencia se ha introducido correctamente
        else:
            print("Sentencia introducida correctamente en remoto")
        #Cerramos el curso
            #cursor.close()
        # Al recibir este comando escribimos en la base de datos local para tener una copia
        try:
        #mandamos directamente el diccionario donde estan almacenados los errores.
        #Los valores de errores, se mandan tal cual son recibidos de la Mega
            cursor_localhost.execute( sql, (0, 0,0,0, 0,0,0, 0,0,0 ))
        #Si hay algun error en la conexion devolvemos un error
        except pymysql.DatabaseError as e:
            print("Error %d: %s" % (e.args[0], e.args[1]))
        # Si no hemos podido conectar a la BBDD al inicio:
        except NameError:
```

```
            print("No se pudo conectar al  base de datos local en el inicio")
        except:
            print("Se ha producido un error al insertar la
                       sentencia en la base de datos local")
        # Si la sentencia se ha introducido correctamente
        else:
            print("Sentencia introducida correctamente en local")
        #Cerramos el curso r
            #cursor_localhost.close()
    ########################################################
        try:
            db_localhost.close()
        except:
            print("[X] Error al cerrar base de datos local")
        else:
            print("[OK] Desconectado de la base de datos local")
        try:
            db.close()
        except:
            print("[X] Error al cerrar base de datos remota")
        else:
            print("[OK] Desconectado de la base de datos remota")
        try:
            mega.close()
        except:
            print("[X] Error al desconectar de la placa MEGA")
        else:
            print("[OK] Desconexión correcta de la placa MEGA")
        print("[OK] Mandado comando de apagado")
        subprocess.check_call("sudo halt",shell=True)
        exit()
    elif (comando[:4] == "FZUM"):
        comando=""
        zum.close()

        status = funcionPuertos()
        if status == 1:
            try:
                cursor_localhost.execute(sql,(62,62,62,62,62,62,62,62,62,62))
            except:
                print("Error guardar 62")
        elif status == 2:
```

```
                try:
                    cursor_localhost.execute(sql,(72,72,72,72,72,72,72,72,72,72))
                except:
                    print("Error guardar 72")
            elif status == 3:
                try:
                    cursor_localhost.execute(sql,(82,82,82,82,82,82,82,82,82,82))
                except:
                    print("Error guardar 82")


          try:
                subprocess.check_call("avrdude -patmega328p -carduino -P/dev/zowi -b 115200 -D
                                       -Uflash:w:/home/pi/zowi/python/ZOWI_BASE_v0.hex:i",
                                       shell=True)
            except subprocess.CalledProcessError:
              try:
                  cursor_localhost.execute(sql,(5,5,5,5,5,5,5,5,5,5))
              except:
                  print("Error guardar 5")
              print("Programacion demo no correcta")
              mega.flushInput()
              mega.flushOutput()
              mega.write("M")
            else:
                print("Programacion demo correcta")
                mega.write("B")
        else:
            try:
                zum.write(comando)
            except NameError:
                print("conexion no establecida")
                #sleep(2)
            else:
                print("dato mandado")
        comando = ""
# Se pulsa ctrl+c
except KeyboardInterrupt:
    print("Interrupción detectada de usuario")
###
    ##################################################
    # Escribimos en las bases de datos un mensaje de fin
```

```
    try:
    # mandamos directamente el diccionario donde están almacenados los errores.
    # Los valores de errores, se mandan tal cual son recibidos de la Mega
       cursor.execute( sql, (0, 0,0,0, 0,0,0, 0,0,0))
    # Si hay algun error en la conexion devolvemos un error
    except pymysql.DatabaseError as e:
       print("Error %d: %s" % (e.args[0], e.args[1]))
    # Si no hemos podido conectar a la BBDD al inicio:
    except NameError:
       print("No se pudo conectar al  base de datos remota en el inicio")
    except:
       print("Se ha producido un error al insertar la sentencia en la base de datos remota")
    # Si la sentencia se ha introducido correctamente
    else:
       print("Sentencia introducida correctamente en remoto")
    #Cerramos el curso
        #cursor.close()
    # Al recibir este comando escribimos en la base de datos local para tener una copia
    try:
        #mandamos directamente el diccionario donde estan almacenados los errores.
    #Los valores de errores, se mandan tal cual son recibidos de la Mega
       cursor_localhost.execute( sql, (0, 0,0,0, 0,0,0, 0,0,0 ))
    #Si hay algun error en la conexion devolvemos un error
    except pymysql.DatabaseError as e:
        print("Error %d: %s" % (e.args[0], e.args[1]))
    # Si no hemos podido conectar a la BBDD al inicio:
    except NameError:
        print("No se pudo conectar al  base de datos local en el inicio")
    except:
        print("Se ha producido un error al insertar la sentencia en la base de datos local")
    # Si la sentencia se ha introducido correctamente
    else:
       print("Sentencia introducida correctamente en local")
    #Cerramos el curso r
                #cursor_localhost.close()
     ########################################################
###
    try:
        db_localhost.close()
    except:
        print("[X] Error al cerrar base de datos local")
    else:
```

```
        print("[OK] Desconectado de la base de datos local")
    try:
        db.close()
    except:
        print("[X] Error al cerrar base de datos remota")
    else:
        print("[OK] Desconectado de la base de datos remota")
    try:
        mega.close()
    except:
        print("[X] Error al desconectar de la placa MEGA")
    else:
        print("[OK] Desconexión correcta de la placa MEGA")
    exit()
except serial.serialutil.SerialException:
    print("Mega desconectada de forma incorreta")
    try:
       cursor_localhost.execute(sql,(2,2,2,2,2,2,2,2,2,2))
    except:
      print("Error escritura 2 en base de datos")
    try:
        db_localhost.close()
    except:
        print("[X] Error al cerrar base de datos local")
    else:
        print("[OK] Desconectado de la base de datos local")
    try:
        db.close()
    except:
        print("[X] Error al cerrar base de datos remota")
    else:
        print("[OK] Desconectado de la base de datos remota")
    try:
        mega.close()
    except:
        print("[X] Error al desconectar de la placa MEGA")
    else:
        print("[OK] Desconexión correcta de la placa MEGA")
    exit()
except:
    print("Otros errores")
    try:
```

```
    cursor_localhost.execute(sql,(3,3,3,3,3,3,3,3,3,3))
except:
   print("Error escritura 3 en base de datos")
exit()
```