Partie code :

Voici le code de notre interface graphique :

import tkinter as tk

from tkinter import messagebox

from tkinter import ttk

import serial

def envois(data):

# Fonction pour envoyer les données au périphérique connecté

ser = serial.Serial(port="COM5", baudrate=9600, timeout=1)

ser.write(bytes(str(data) + '\r\n', 'utf-8'))

ser.flush()

ser.close()

def update\_distance\_limit():

# Fonction pour mettre à jour la limite de distance

limit = distance\_limit\_entry.get()

try:

limit = int(limit)

if 1 <= limit <= 250:

limite\_var.set(limit)

else:

raise ValueError

except ValueError:

messagebox.showerror(

"Erreur", "Veuillez entrer une valeur numérique entière entre 1 et 250 pour la limite de distance.")

def on\_distance\_limite\_send(event):

# Fonction appelée lors de l'envoi de la limite de distance

on\_distance\_change(event)

data = str(int(distance\_var.get())) + "," + str(int(limite\_var.get()))

envois(data)

def on\_distance\_change(event):

# Fonction appelée lors du changement de la distance

distance = int(distance\_scale.get())

distance\_var.set(distance)

# Création de la fenêtre principale

window = tk.Tk()

window.title("Interface de distance")

window.geometry("500x400") # Taille de la fenêtre

# Variable pour stocker la valeur de distance

distance\_var = tk.IntVar(value=1)

# Variable pour stocker la valeur de limite de distance

limite\_var = tk.IntVar(value=250)

# Création des éléments de l'interface utilisateur

title\_label = ttk.Label(window, text="Distance :", font=("Arial", 12, "bold"))

title\_label.pack(pady=10)

distance\_scale = ttk.Scale(window, from\_=0, to=250,

length=250, orient=tk.HORIZONTAL)

distance\_scale.pack(pady=10)

distance\_scale.set(100)

distance\_limit\_label = ttk.Label(

window, text="Limite de distance :", font=("Arial", 12, "bold"))

distance\_limit\_label.pack()

distance\_limit\_entry = ttk.Entry(window)

distance\_limit\_entry.pack(pady=10)

update\_button = ttk.Button(

window, text="Mettre à jour", command=update\_distance\_limit)

update\_button.pack(pady=10)

ttk.Label(window, text="Distance :", font=("Arial", 12, "bold")).pack()

distance\_text\_label = ttk.Label(

window, textvariable=distance\_var, font=("Arial", 14, "bold"))

distance\_text\_label.pack()

ttk.Label(window, text="Limite à ne pas dépasser :",

font=("Arial", 12, "bold")).pack()

distance\_text\_label = ttk.Label(

window, textvariable=limite\_var, font=("Arial", 14, "bold"))

distance\_text\_label.pack()

distance\_scale.bind("<B1-Motion>", on\_distance\_change)

distance\_scale.bind("<ButtonRelease-1>", on\_distance\_limite\_send)

window.mainloop()

Ce code n’est pas stocké dans le Raspberry mais bien sur le pc et est exécutable depuis le pc. Il est utilisé pour communiquer avec le Raspberry. C’est grâce à cette fenêtre que nous simulons la sonde et que nous adaptons la limite de distance.

Voici le code du Raspberry :

import machine

import utime

import sys

import select

poller = select.poll()

poller.register(sys.stdin, 1)

alarm = 250

distance = 99

# Fonction pour mesurer la distance avec la sonde HC-SR04

def read\_from\_port():

if poller.poll(0):

line = sys.stdin.buffer.readline()

if line:

data = line.decode('utf-8')

return data

# Fonction pour mesurer la distance avec la sonde HC-SR04

def dechiffre():

global distance, alarm

data = read\_from\_port()

if data is not None:

data\_Array = data.split(',')

distance = int(data\_Array[0])

alarm = int(data\_Array[1])

def alarme():

red\_led.on()

green\_led.off()

# Définition des chiffres à afficher (en binaire)

digit\_patterns = {

0: [0, 0, 0, 0, 0, 0, 1],

1: [1, 0, 0, 1, 1, 1, 1],

2: [0, 0, 1, 0, 0, 1, 0],

3: [0, 0, 0, 0, 1, 1, 0],

4: [1, 0, 0, 1, 1, 0, 0],

5: [0, 1, 0, 0, 1, 0, 0],

6: [0, 1, 0, 0, 0, 0, 0],

7: [0, 0, 0, 1, 1, 1, 1],

8: [0, 0, 0, 0, 0, 0, 0],

9: [0, 0, 0, 0, 1, 0, 0]

}

# Définition des broches pour les deux afficheurs 7 segments

afficheur\_Dixaines = [

machine.Pin(8, machine.Pin.OUT),

machine.Pin(9, machine.Pin.OUT),

machine.Pin(10, machine.Pin.OUT),

machine.Pin(11, machine.Pin.OUT),

machine.Pin(12, machine.Pin.OUT),

machine.Pin(13, machine.Pin.OUT),

machine.Pin(14, machine.Pin.OUT)

]

afficheur\_Unitees = [

machine.Pin(16, machine.Pin.OUT),

machine.Pin(20, machine.Pin.OUT),

machine.Pin(21, machine.Pin.OUT),

machine.Pin(22, machine.Pin.OUT),

machine.Pin(26, machine.Pin.OUT),

machine.Pin(27, machine.Pin.OUT),

machine.Pin(28, machine.Pin.OUT)

]

# Définition de la broche pour le point

dot\_point = machine.Pin(5, machine.Pin.OUT)

# Définition des broches pour les leds

green\_led = machine.Pin(1, machine.Pin.OUT)

red\_led = machine.Pin(0, machine.Pin.OUT)

red\_led.off()

green\_led.on()

# Définition de la broche pour la sonde HC-SR04

trigger = machine.Pin(18, machine.Pin.OUT)

echo = machine.Pin(19, machine.Pin.IN)

# Fonction pour afficher un chiffre sur un afficheur 7 segments

def afficher\_chiffre(digit, segments):

digit = int(digit)

for i in range(len(segments)):

if digit\_patterns[digit][i]:

segments[i].on()

else:

segments[i].off()

# Boucle principale pour mesurer la distance et afficher les deux premiers chiffres

while True:

# Mesure la distance

dechiffre()

if distance > alarm:

alarme()

else:

red\_led.off()

green\_led.on()

machine.Pin(8, machine.Pin.OUT).on()

if distance > 99:

dot\_point.off()

else:

dot\_point.on()

dist = str(distance)

# Récupère les deux premiers chiffres de la distance

if distance < 10:

for segment in afficheur\_Dixaines:

segment.on()

second\_digit = dist[0]

afficher\_chiffre(second\_digit, afficheur\_Unitees)

else :

first\_digit = dist[0]

second\_digit = dist[1]

afficher\_chiffre(first\_digit, afficheur\_Dixaines)

afficher\_chiffre(second\_digit, afficheur\_Unitees)

# # Attend une seconde avant de mesurer à nouveau

utime.sleep(0.5)

Ce code est utilisé pour afficher la distance et déclencher un led lorsque l’alarme est déclenché. C’est ce code qui fait fonctionner le projet en grande partie et qui est stocké depuis le Raspberry directement.