

# Rapport : Traitement du signal




# Sommaire :

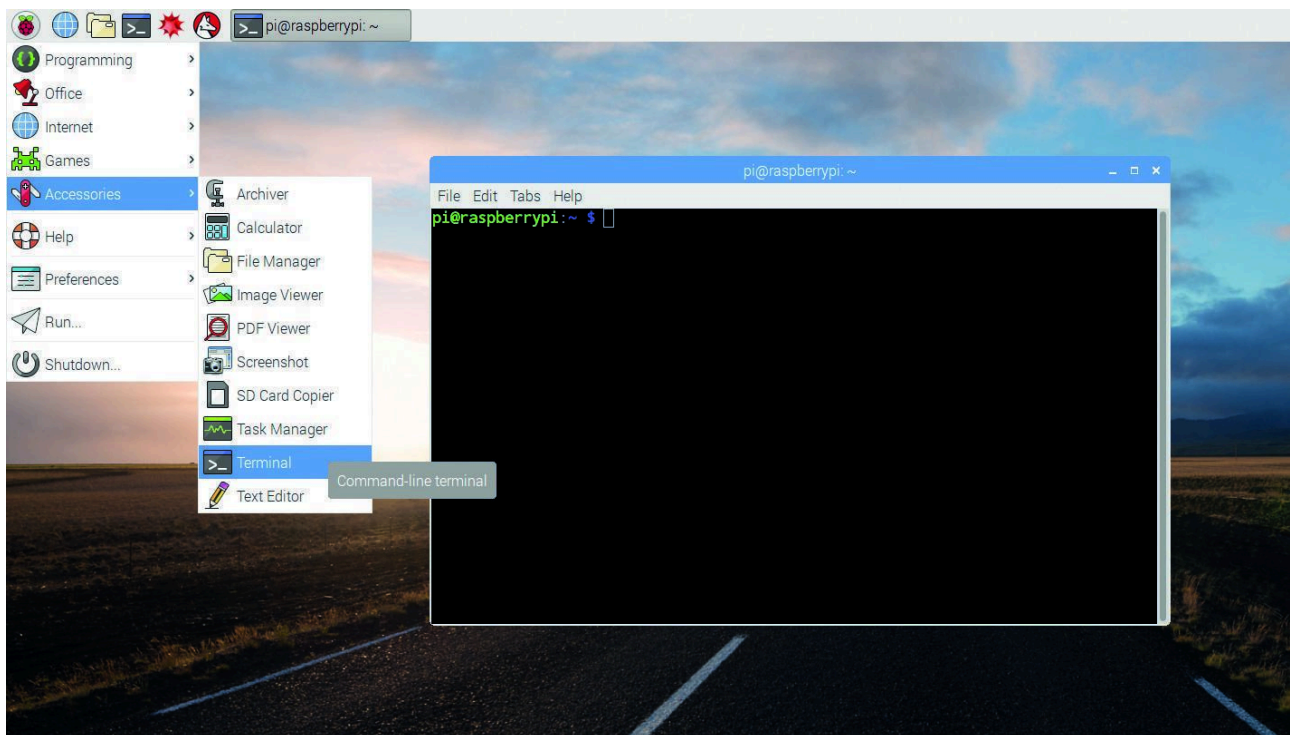
Raspberry pi OS :	3
Installer Mosquitto	4
<b>Partie Ultrason</b>	<b>5</b>
Trouver un capteur ultrason	5
Exemple de branchement des capteurs	5
Exemple de programme python	6
Exemple de simulation :	7
<b>Partie son</b>	<b>8</b>
Trouver un microphone	8
Câblage du microphone	9
Exemple de code du microphone	11
Script final du capteur sonore	13
Vue d'ensemble du script	19

## Raspberry pi OS :

La première étape est d'installer « Raspberry PI OS », afin d'avoir la garantie que tout est optimisé et compatible avec votre Raspberry Pi.



Une fois sur l'os de Raspberry PI, il faut faire la mise à jour des paquets s'effectue via quelques lignes de commandes à exécuter dans un terminal. 



Enfin exécuter ces 2 commandes :

- `sudo apt-get update` // Mise à jour des informations des paquets disponibles pour la distribution.
- `sudo apt-get dist-upgrade` // Mise à jour complète de tous les paquets.

## Installer Mosquitto

En ligne de commande il faut exécuter ces 2 commandes :

- `sudo apt install mosquitto mosquitto-clients -y`
- `sudo systemctl enable mosquitto`

Ces 2 commandes font en sorte que le broker soit actif et tourne sur localhost (port 1883 par défaut).

Ensuite il faut tester Mosquitto en local :

- `Mosquitto_sub -h «default localhost» -t -v // Créer un abonnement`

Ensuite dans un autre terminal, faire :

- `Mosquitto_pub -t « topic » -m « message » //Envoie un message`

Envoie de données à un broker MQTT

Ajouter dans le script :

- `client.connect(« IP_Du_Broker”, 1883, 60)`

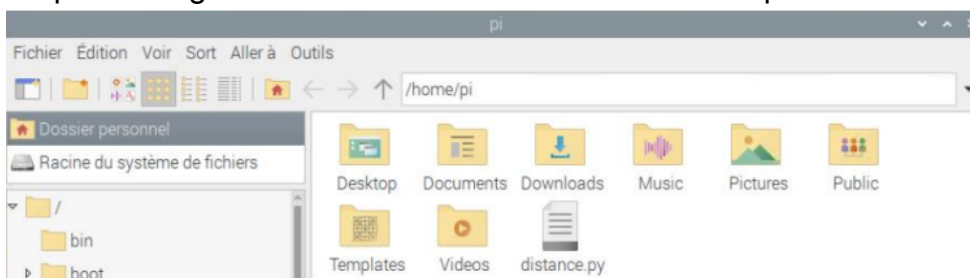
## Python

Afin de faire fonctionner nos différents programmes sur la Raspberry PI on va utiliser Python

⚠ Si il n'y a pas python sur l'OS, faire dans un terminal : `sudo apt install python3-pip`

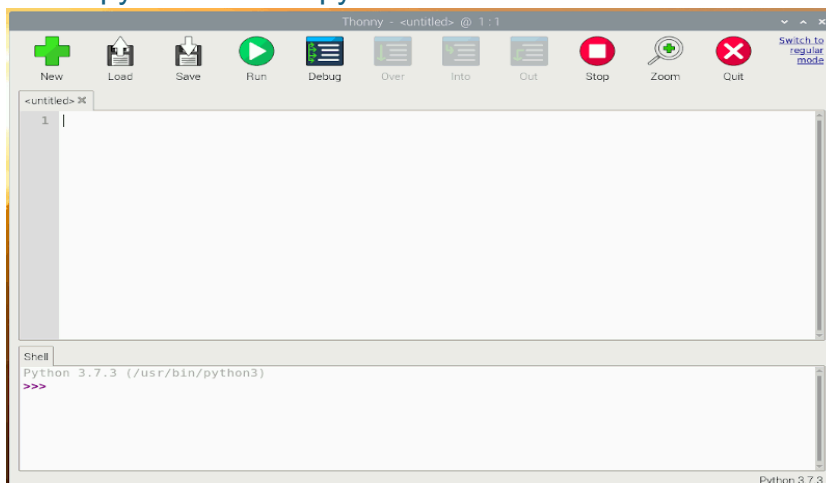
⚠ Ne pas oublier d'installer la bibliothèque MQTT : `pip3 install paho-mqtt`

On peut enregistrer nos codes dans le dossier /home/pi



Puis dans le terminal exécuter la commande :

- `sudo python3 sae24.py // ouvre un IDE`



# Partie Ultrason

## Trouver un capteur ultrason

Le premier capteur ultrason auquel nous avons pensé est le HC-SR24 , il est très populaire et facile à utiliser, car on trouve plein de tutoriels sur Internet. Mais le problème, c'est qu'il a une portée de seulement 4 mètres.

### Caractéristiques:

- Alimentation: 3,3 ou 5 Vcc
- Consommation: 15 mA
- Fréquence: 40 kHz
- Portée: de 2 cm à 4 m
- Déclenchement: impulsion TTL positive de 10µs
- Signal écho: impulsion positive TTL proportionnelle à la distance.
- Calcul: distance (cm) = impulsion (µs) / 58
- Trous de fixation: 1,8 mm
- Dimensions: 45 x 20 x 18 mm

SRC: <https://www.gotronic.fr/>

Malgré le fait que sa portée soit de seulement 4 mètres, on va quand même l'utiliser comme capteur pour notre simulation.

## Exemple de branchement des capteurs

Les branchements des capteurs ultrasons ont été faits à l'aide d'une documentation trouvée sur Internet.

Source :

<https://github.com/KryptoWard/SA-24/blob/main/Traitement%20du%20signal/pj2-guide-us-hc-sr04-raspberry-pi-2310.pdf>

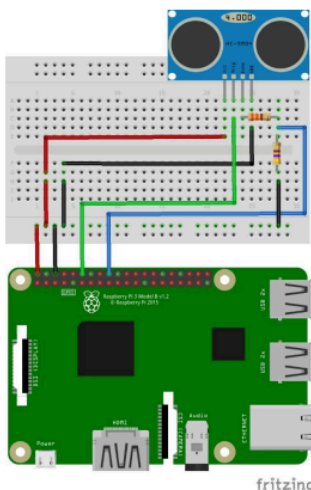


Tableau de correspondance :

Raspberry Pi	HC-SR04
5 V	Vcc
GND	Gnd
GPIO24	Echo*
GPIO18	Trig

\* La sortie Echo comporte un diviseur de tension basé sur une résistance de 330 Ω (orange, orange, marron et or) et une de 470 Ω (jaune, violet, marron et or).

Ce pont diviseur permet d'atténuer la tension de 5 Vcc en sortie du capteur afin de ne pas endommager l'entrée digitale de la carte Raspberry Pi (uniquement compatible 3,3 Vcc).

Sur ce branchement, les résistances sont là pour éviter toute surcharge de tension.

Sur le capteur ultrason il y a 4 pins, tension, echo, trigger et ground. Tension est celui qui reçoit l'électricité, ground correspond à la masse, trigger envoie les ondes, et echo réceptionne les données.

## Exemple de programme python

Exemple de programme d'un capteur à ultrasons HC-SR24.

Source :

<https://github.com/KryptoWard/SA-24/blob/main/Traitement%20du%20signal/pj2-guide-us-hc-sr04-raspberry-pi-2310.pdf>

```
# Import des librairies GPIO et time (temps et conversion) #
import RPi.GPIO as GPIO
import time

# Module GPIO: BOARD ou BCM (numérotation comme la sérigraphie de la carte ou comme le chip) #
GPIO.setmode(GPIO.BCM)

# Définition des broches GPIO #
GPIO_TRIGGER = 18
GPIO_ECHO = 24

# Définition des broches en entrée ou en sortie #
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # Mise à l'état haut de la broche Trigger #
    GPIO.output(GPIO_TRIGGER, True)

    # Mise à l'état bas de la broche Trigger après 10 µs #
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # Enregistrement du temps de départ des ultrasons #
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # Enregistrement du temps d'arrivée des ultrasons #
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # Calcul de la durée de l'aller-retour des US #
    TimeElapsed = StopTime - StartTime
    # On multiplie la durée par la vitesse du son: 34300 cm/s #
    # Et on divise par deux car il s'agit d'un aller et retour. #
    distance = (TimeElapsed * 34300) / 2

    return distance

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Distance mesurée = %.1f cm" % dist)
            time.sleep(1)

        # On reset le programme via CTRL+C #
    except KeyboardInterrupt:
        print("Mesure stoppée")
        GPIO.cleanup()
```

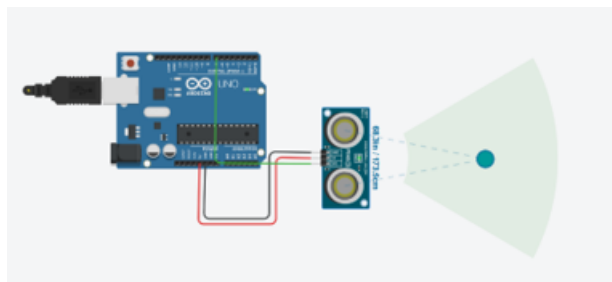
Commentaire :

Dans la vraie vie, il faut faire attention à la broche à laquelle ont à relier TRIGGER et ECHO, ce n'est pas forcément 18 et 24 sur tous les capteurs.

De plus, le HC-SR04 fonctionne avec du 5V donc si on utilise un microcontrôleur 3.3V(ESP32), il faut abaisser le niveau logique du signal ECHO avec un diviseur de tension.

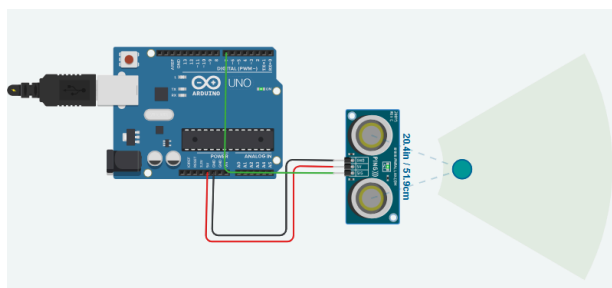
## Exemple de simulation :

Simulation d'un capteur ultrasons Arduino UNO sur « TINKERCARD ».



### Moniteur série

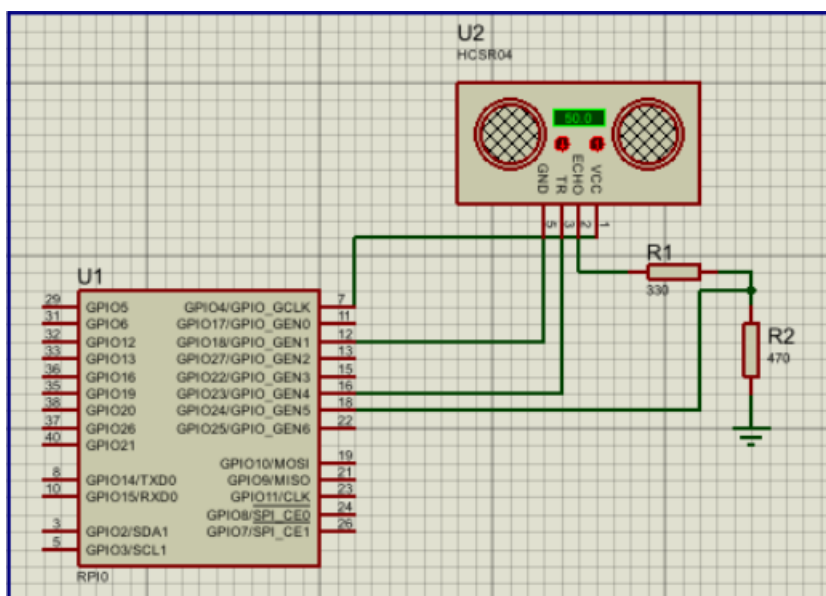
```
72in, 183cm
72in, 183cm
72in, 183cm
72in, 183cm
72in, 183cm
```



### Moniteur série

```
20in, 51cm
20in, 51cm
20in, 52cm
20in, 51cm
20in, 51cm
```

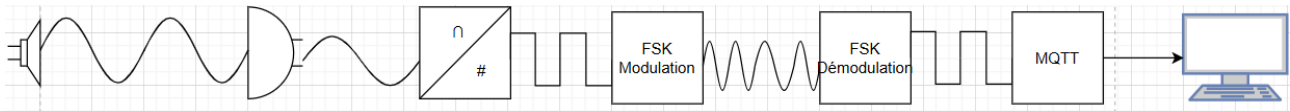
Simulation d'un capteur ultrasons HC-SR24 et d'une Raspberry PI sur « PROTEUS ».



À noter que la simulation « Proteus » ne fonctionne pas, car n'ayant pas les licences pour la Raspberry PI et la HC-SR24, le logiciel ne nous permet pas d'effectuer la simulation à part si on achète la licence « Proteus Professional ».

# Partie son

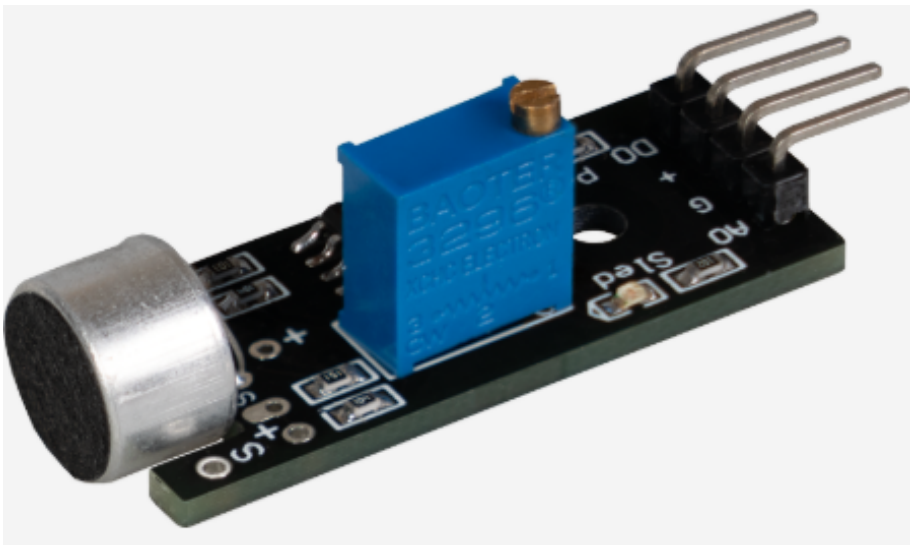
Schéma des différentes étapes entre le capteur et la récupération de données sur le serveur MQTT.



## Trouver un microphone

Le 1<sup>er</sup> objectif de la partie son, c'est de trouver un microphone qui est compatible avec un Raspberry Pi.

Donc, après avoir cherché sur Internet, on a trouvé un micro analogique (KY-037).



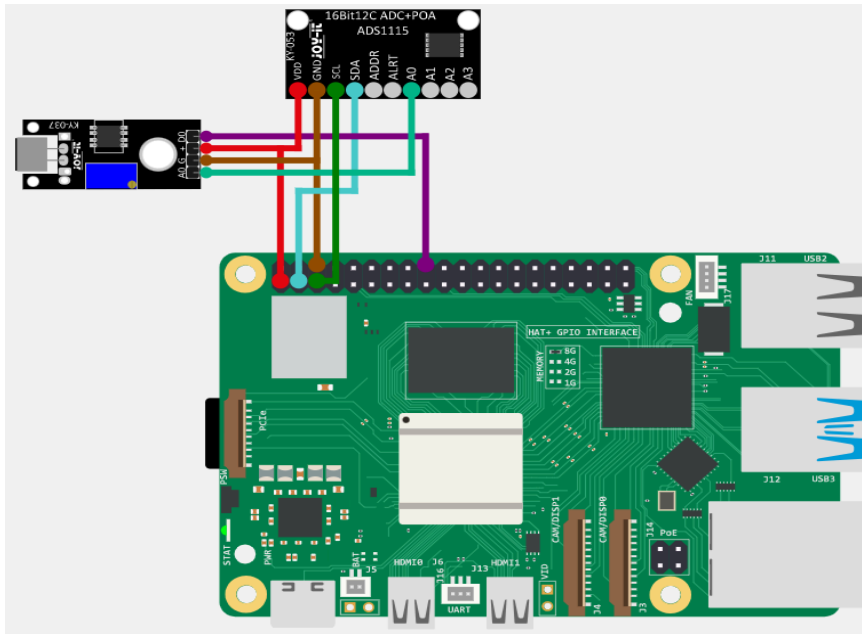
De plus, on a trouvé un site internet qui montre l'installation du microphone sur une Raspberry.

Source: <https://sensorkit.joy-it.net/fr/sensors/ky-037>



## Câblage du microphone

Sur le site internet on peut voir ce schéma :



Contrairement à l'Arduino, le Raspberry Pi n'a pas d'entrées analogiques qui convertit les données en numérique à l'aide d'un CAN. Donc si on veut utiliser des capteurs où les valeurs de sortie sont numériques, il ne faut pas utiliser un Raspberry Pi.

Le capteur son ne fonctionne pas avec un bouton on/off, il est équipé d'un potentiomètre qui ajuste sa sensibilité.

Convertisseur analogique numérique :

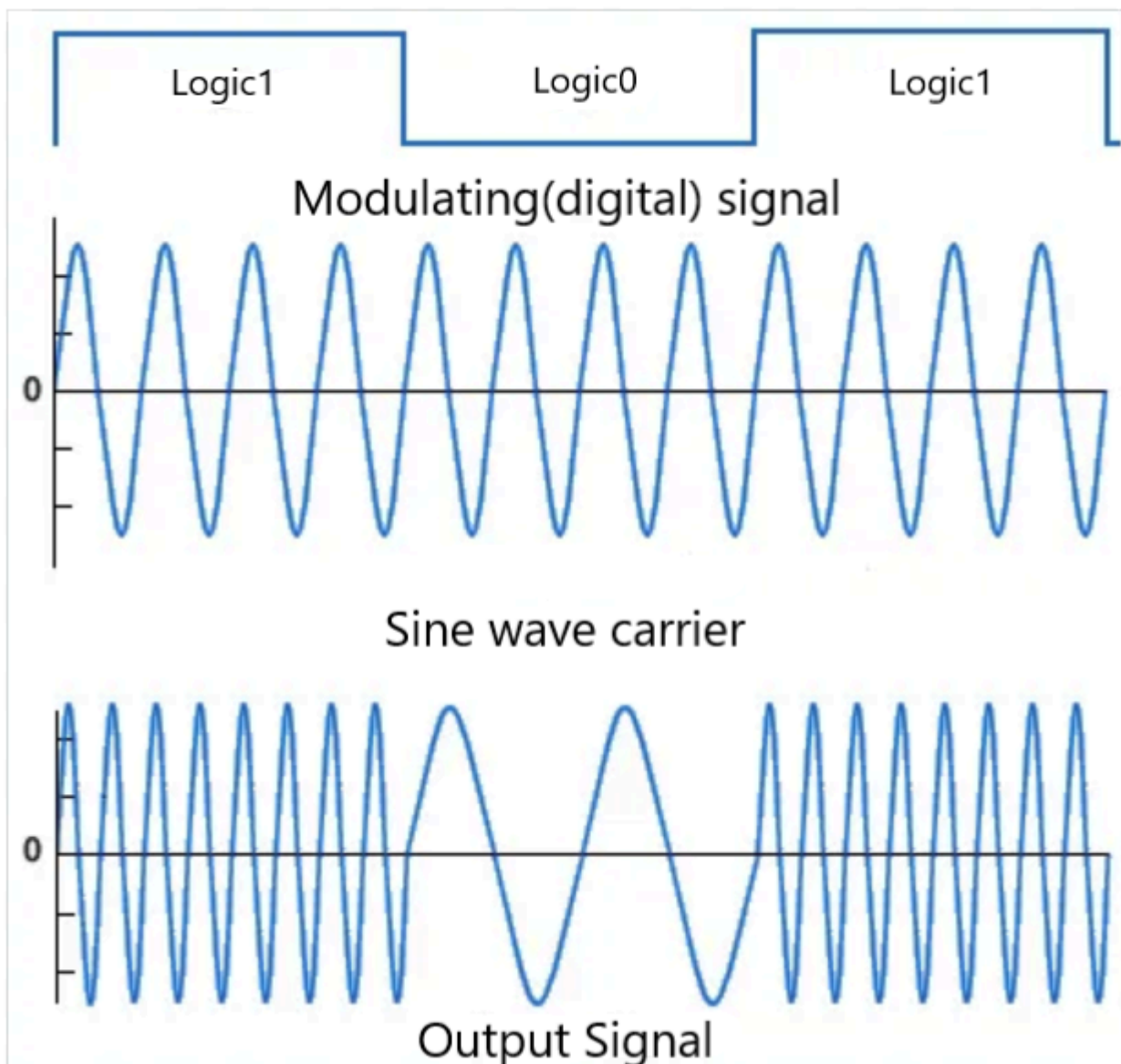
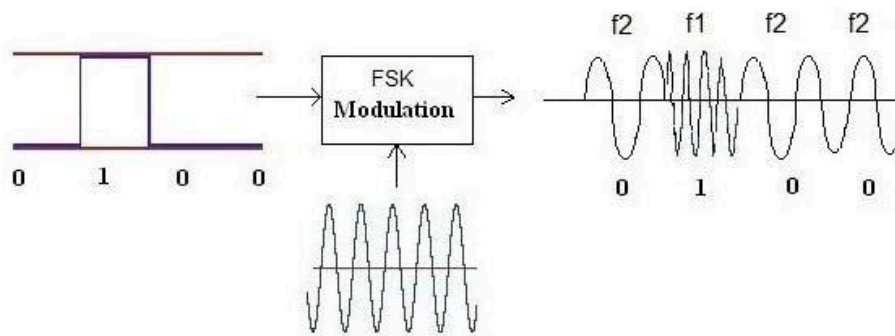


### **Le rôle du FSK (Frequency Shift Keying)**

Le FSK est une technique de modulation numérique où l'information est encodée en faisant varier la fréquence du signal porteur. Dans le contexte de ton capteur sonore :

- **Principe** : Le capteur émet des signaux à différentes fréquences selon la distance de l'objet détecté
- **Avantage** : Plus robuste aux interférences que la modulation d'amplitude, car les variations de fréquence sont moins sensibles au bruit
- **Application** : Permet de distinguer différents types d'objets ou différentes distances en fonction des fréquences retournées

## FSK Modulation | Frequency Shift Keying Modulation



Source : <https://www.fr-ebyte.com/news/559>

## **Exemple de code du microphone**

Programme du fabricant du capteur sonore :

```

from Adafruit_ADS1x15 import ADS1x15
from time import sleep
# import needed modules
import math, signal, sys, os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
# initialise variables
delayTime = 0.5 # in Sekunden
# assigning the ADS1x15 ADC
ADS1015 = 0x00 # 12-bit ADC
ADS1115 = 0x01 # 16-bit
# choosing the amplifying gain
gain = 4096 # +/- 4.096V
# gain = 2048 # +/- 2.048V
# gain = 1024 # +/- 1.024V
# gain = 512 # +/- 0.512V
# gain = 256 # +/- 0.256V
# choosing the sampling rate
# sps = 8 # 8 Samples per second
# sps = 16 # 16 Samples per second
# sps = 32 # 32 Samples per second
sps = 64 # 64 Samples per second
# sps = 128 # 128 Samples per second
# sps = 250 # 250 Samples per second
# sps = 475 # 475 Samples per second
# sps = 860 # 860 Samples per second
# assigning the ADC-Channel (1-4)
adc_channel_0 = 0 # Channel 0
adc_channel_1 = 1 # Channel 1
adc_channel_2 = 2 # Channel 2
adc_channel_3 = 3 # Channel 3
KY-037 Microphone sensor module (high sensitivity)
Page
of
6
7
Export: 02.03.2021
This document was created with BlueSpice
adc_channel_3 = 3 # Channel 3
# initialise ADC (ADS1115)
adc = ADS1x15(ic=ADS1115)
# Input pin for the digital signal will be picked here
Digital_PIN = 24
GPIO.setup(Digital_PIN, GPIO.IN, pull_up_down = GPIO.PUD_OFF)
#####
#####

```

```

#####
# main program loop
#####
# The program reads the current value of the input pin
# and shows it at the terminal
try:
    while True:
        #Current values will be recorded
        analog = adc.readADCSingleEnded(adc_channel_0, gain, sps)
        # Output at the terminal
        if GPIO.input(Digital_PIN) == False:
            print "Analog voltage value:", analog,"mV, ", "extreme value: not reached"
        else:
            print "Analog voltage value:", analog, "mV, ", "extreme value: reached"
            print "-----"
            sleep(delayTime)
except KeyboardInterrupt:
    GPIO.cleanup()

```

Programme du site <https://sensorkit.joy-it.net/fr/sensors/ky-037> :

```

import time
import board
import busio
import adafruit_ads1x15.ads1115 as ADS
from adafruit_ads1x15.analog_in import AnalogIn
from gpiozero import DigitalInputDevice

# Créer le bus I2C
i2c = busio.I2C(board.SCL, board.SDA)

# Créer l'objet ADC avec le bus I2C
ads = ADS.ADS1115(i2c)

# Créer des entrées asymétriques sur les canaux
chan0 = AnalogIn(ads, ADS.P0)
chan1 = AnalogIn(ads, ADS.P1)
chan2 = AnalogIn(ads, ADS.P2)
chan3 = AnalogIn(ads, ADS.P3)

delayTime = 1
# Initialisation de l'appareil d'entrée numérique pour le capteur au niveau du GPIO 24
digital_pin = DigitalInputDevice(24, pull_up=False) # pull_up=False, parce que
pull_up_down=GPIO.PUD_OFF

while True:
    analog = '%.2f' % chan0.voltage

    # Sortie vers la console
    if not digital_pin.is_active:
        print("Valeur de la tension analogique:", analog, "V, ", "Valeur limite : pas encore

```

```

atteinte")
    else:
        print("Valeur de la tension analogique:", analog, "V, ", "Valeur limite : atteinte")
        print("-----")

    # Réinitialisation + délai
    time.sleep(delayTime)

```

## **Script final du capteur sonore**

```

import time
import board
import busio
import adafruit_ads1x15.ads1115 as ADS
from adafruit_ads1x15.analog_in import AnalogIn
from gpiozero import DigitalInputDevice
import math
import paho.mqtt.client as mqtt
import json

# Créer le bus I2C
i2c = busio.I2C(board.SCL, board.SDA)

# Créer l'objet ADC avec le bus I2C
ads = ADS.ADS1115(i2c)

# Créer des entrées asymétriques sur les canaux
chan0 = AnalogIn(ads, ADS.P0) # Canal principal pour le capteur sonore
chan1 = AnalogIn(ads, ADS.P1)
chan2 = AnalogIn(ads, ADS.P2)
chan3 = AnalogIn(ads, ADS.P3)

# CONFIGURATION MQTT
MQTT_BROKER = "SAE24_simulation" # Nom ou IP du broker
MQTT_PORT = 1883 # Port standard MQTT
MQTT_TOPIC_DETECTION = "capteur/detection" # Topic pour les détections
MQTT_TOPIC_STATUS = "capteur/status" # Topic pour le statut
MQTT_CLIENT_ID = "capteur_sonore_01" # ID unique du client

# Variables MQTT
mqtt_client = None
mqtt_connected = False
# Choisir la configuration de votre pièce (décommenter la ligne appropriée)
# CONFIGURATION DE LA PIÈCE
# Choisir la configuration de votre pièce (décommenter la ligne appropriée)
LARGEUR_PIECE = 8.0 # Pour 64m² : 8m x 8m
LONGUEUR_PIECE = 8.0
# LARGEUR_PIECE = 0.5 # Pour 0.25m² : 0.5m x 0.5m
# LONGUEUR_PIECE = 0.5

# Position du capteur dans le coin (0,0)
POSITION_CAPTEUR_X = 0.0

```

```

POSITION_CAPTEUR_Y = 0.0

# Calcul des distances caractéristiques de la pièce
DISTANCE_MAX = math.sqrt(LARGEUR_PIECE**2 + LONGUEUR_PIECE**2) #
Diagonale
DISTANCE_CENTRE = math.sqrt((LARGEUR_PIECE/2)**2 +
(LONGUEUR_PIECE/2)**2) # Vers le centre
DISTANCE_COIN_OPPOSE = DISTANCE_MAX # Coin opposé

# Configuration des seuils basés sur les dimensions de la pièce
# Principe : Plus la tension est élevée, plus l'objet est proche
TENSION_MAX_CAPTEUR = 5.0 # Tension maximale du capteur (à ajuster selon votre
matériel)
SEUIL_DETECTION_MIN = 0.05 # Seuil minimum pour détecter un objet

# Calcul des seuils adaptatifs basés sur la géométrie de la pièce
def calculer_seuils():
    """
    Calcule les seuils de tension basés sur les distances caractéristiques
    Utilise le principe : amplitude  $\propto 1/\text{distance}^2$ 
    """
    # Coefficient de calibration (à ajuster selon votre capteur)
    K_CALIBRATION = 2.0

    seuils = {
        'tres_proche': K_CALIBRATION * (1 / (0.5**2)),    # Moins de 50cm
        'proche': K_CALIBRATION * (1 / (1.0**2)),        # Moins de 1m
        'moyen': K_CALIBRATION * (1 / (2.0**2)),         # Moins de 2m
        'centre': K_CALIBRATION * (1 / (DISTANCE_CENTRE**2)), # Vers le centre
        'loin': K_CALIBRATION * (1 / (DISTANCE_MAX**2))  # Diagonale max
    }

    # Normalisation des seuils pour rester dans la plage du capteur
    max_seuil = max(seuils.values())
    if max_seuil > TENSION_MAX_CAPTEUR:
        facteur = TENSION_MAX_CAPTEUR / max_seuil
        for key in seuils:
            seuils[key] *= facteur

    return seuils

SEUILS = calculer_seuils()

delayTime = 0.3

# Initialisation GPIO
digital_pin = DigitalInputDevice(24, pull_up=False)

# Buffer pour lisser les mesures
buffer_size = 7
voltage_buffer = []

```

```

def estimer_distance(voltage):
    """
    Estime la distance basée sur la tension et la géométrie de la pièce
    Utilise la formule inverse : distance  $\approx \sqrt{K/tension}$ 
    """
    if voltage <= SEUIL_DETECTION_MIN:
        return None

    # Coefficient de calibration (ajustable selon le capteur)
    K_DISTANCE = 2.0
    distance_estimee = math.sqrt(K_DISTANCE / voltage)

    # Limite la distance à la diagonale maximale de la pièce
    return min(distance_estimee, DISTANCE_MAX)

def analyser_position(voltage):
    """
    Analyse la position de l'objet dans la pièce basée sur la tension
    """
    if voltage < SEUIL_DETECTION_MIN:
        return "Aucun objet détecté"

    if voltage >= SEUILS['tres_proche']:
        return "Objet à moins de 50 cm"
    elif voltage >= SEUILS['proche']:
        return "Objet à moins d'un mètre"
    elif voltage >= SEUILS['moyen']:
        return "Objet à moins de deux mètres"
    elif voltage >= SEUILS['centre']:
        return "Objet à moins de trois mètres"
    else:
        return "Objet détecté au-delà de trois mètres"

def lisser_mesure(nouvelle_valeur):
    """
    Lisse les mesures pour éviter les variations erratiques
    """
    global voltage_buffer

    voltage_buffer.append(nouvelle_valeur)
    if len(voltage_buffer) > buffer_size:
        voltage_buffer.pop(0)

    return sum(voltage_buffer) / len(voltage_buffer)

# FONCTIONS MQTT
def on_connect(client, userdata, flags, rc):
    """Callback appelé lors de la connexion au broker MQTT"""
    global mqtt_connected
    if rc == 0:

```

```
mqtt_connected = True
print("Connexion MQTT réussie")
# Publier un message de statut
publier_status("ONLINE")
else:
    mqtt_connected = False
    print(f" Échec connexion MQTT, code: {rc}")

def on_disconnect(client, userdata, rc):
    """Callback appelé lors de la déconnexion du broker MQTT"""
    global mqtt_connected
    mqtt_connected = False
    print(" Déconnexion MQTT")

def on_publish(client, userdata, mid):
    """Callback appelé après publication d'un message"""
    pass # Optionnel: pour debug

def initialiser_mqtt():
    """Initialise la connexion MQTT"""
    global mqtt_client

    try:
        mqtt_client = mqtt.Client(MQTT_CLIENT_ID)
        mqtt_client.on_connect = on_connect
        mqtt_client.on_disconnect = on_disconnect
        mqtt_client.on_publish = on_publish

        print(f" Connexion au broker MQTT: {MQTT_BROKER}:{MQTT_PORT}")
        mqtt_client.connect(MQTT_BROKER, MQTT_PORT, 60)
        mqtt_client.loop_start() # Démarre la boucle en arrière-plan

        # Attendre la connexion (max 5 secondes)
        for i in range(50):
            if mqtt_connected:
                break
            time.sleep(0.1)

        if not mqtt_connected:
            print(" Timeout connexion MQTT - Continuera en mode local")

    except Exception as e:
        print(f" Erreur initialisation MQTT: {e}")
        mqtt_client = None

def publier_detection(message, tension, timestamp=None):
    """Publie une détection via MQTT"""
    if not mqtt_connected or mqtt_client is None:
        return

    try:
        if timestamp is None:
```



```

        timestamp = time.time()

    # Création du payload JSON
    payload = {
        "message": message,
        "tension": round(tension, 3),
        "timestamp": timestamp,
        "capteur_id": MQTT_CLIENT_ID,
        "piece_config": {
            "largeur": LARGEUR_PIECE,
            "longueur": LONGUEUR_PIECE,
            "surface": LARGEUR_PIECE * LONGUEUR_PIECE
        }
    }

    # Publication
    result = mqtt_client.publish(MQTT_TOPIC_DETECTION, json.dumps(payload))

    if result.rc == mqtt.MQTT_ERR_SUCCESS:
        print(f"MQTT envoyé: {message}")
    else:
        print(f"Erreur envoi MQTT: {result.rc}")

except Exception as e:
    print(f"Erreur publication MQTT: {e}")

def publier_status(status):
    """Publie le statut du capteur"""
    if not mqtt_connected or mqtt_client is None:
        return

    try:
        payload = {
            "status": status,
            "timestamp": time.time(),
            "capteur_id": MQTT_CLIENT_ID
        }

        mqtt_client.publish(MQTT_TOPIC_STATUS, json.dumps(payload))

    except Exception as e:
        print(f"Erreur publication statut: {e}")

# Affichage des informations de configuration
print("=== CAPTEUR SONORE DE POSITIONNEMENT AVEC MQTT ===")
print(f"Configuration de la pièce: {LARGEUR_PIECE}m x {LONGUEUR_PIECE}m")
print(f"Surface: {LARGEUR_PIECE * LONGUEUR_PIECE}m²")
print(f"Distance maximale (diagonale): {DISTANCE_MAX:.2f}m")
print(f"Capteur positionné dans le coin (0,0)")
print(f"Broker MQTT: {MQTT_BROKER}:{MQTT_PORT}")
print(f"Topics: {MQTT_TOPIC_DETECTION}, {MQTT_TOPIC_STATUS}")

```

```
print("\nInitialisation MQTT...")
initialiser_mqtt()

print("Initialisation capteur...")
time.sleep(2)
print("Détection active !\n")

try:
    compteur = 0
    derniere_detection = ""

    while True:
        # Lecture et lissage de la tension
        tension_brute = chan0.voltage
        tension_lissee = lisser_mesure(tension_brute)

        # Analyse de la position
        message_position = analyser_position(tension_lissee)

        # Affichage local
        print(message_position)

        # Envoi MQTT seulement si le message a changé (évite le spam)
        if message_position != derniere_detection:
            publier_detection(message_position, tension_lissee)
            derniere_detection = message_position

        compteur += 1

        # Envoi périodique du statut (toutes les 50 mesures)
        if compteur % 50 == 0:
            publier_status("ACTIVE")
            print(f" Heartbeat envoyé (mesure #{compteur})")

        time.sleep(delayTime)

except KeyboardInterrupt:
    print("\n\nArrêt du programme par l'utilisateur")
    publier_status("OFFLINE")
    print(f"Total de mesures effectuées: {compteur}")
except Exception as e:
    print(f"Erreur: {e}")
    publier_status("ERROR")
finally:
    if mqtt_client:
        mqtt_client.loop_stop()
        mqtt_client.disconnect()
    print("Programme terminé")
```

## **Vue d'ensemble du script**

Ce script transforme un capteur sonore en système de détection de distance avec transmission MQTT. Il utilise un Raspberry Pi avec un convertisseur analogique-numérique pour analyser les signaux sonores et déterminer la distance d'objets.

### **Fonctionnalités principales**

- Acquisition des données
- Calcul adaptatif des seuils
- Classification de distance
- Format des données MQTT
- Robustesse