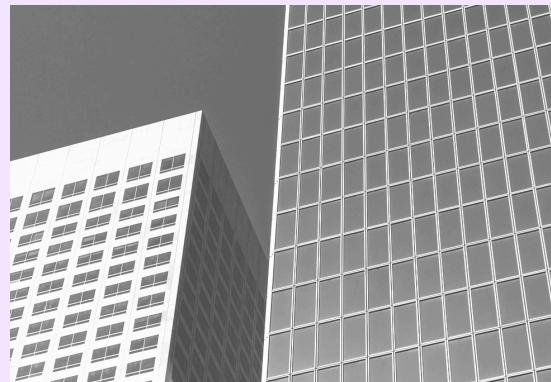




PROJET CASQUE CONNECTÉ



Etudiant 1 : Lilian Bowen

Etudiant 2 : Kyllian Delompré

Etudiant 3 : Pedro Gomes

Table des Matières

Introduction.....	4
Répartition des tâches.....	5
Projet Timeline.....	6
Diagrammes.....	7
1. Diagramme de cas d'utilisation.....	7
2. Diagramme de séquence.....	8
3. Diagramme de Flowchart.....	9
Travail Etudiant n°1.....	10
<u>Composants électroniques.....</u>	10
1. Capteur de luminosité.....	10
2. Accéléromètre.....	13
3. Bouton poussoir.....	15
<u>Programmes Python.....</u>	17
1. Programmation Bouton poussoir.....	17
2. Programmation Accéléromètre.....	20
3. Programmation Capteur de lumière.....	23
4. Bouton poussoir + Accéléromètre.....	28

Travail Etudiant n°2.....	31
1. Convertisseur analogique.....	31
2. Capteur Gaz.....	35
3. Capteur Température et humidité.....	42
<u>Programmes Python Compilation des deux capteurs</u>	47
1. Boucle Principale.....	47
2. Insertion des données dans les bases de données.....	48
3. Gestion des erreurs et attente.....	49
4. Gestion des interruptions et nettoyage.....	49
<u>Programmes Python Envoie des données.....</u>	51
1. Bibliothèques importées.....	51
2. Initialisation des Broches GPIO.....	51
3. Connexion à la base de données MySQL.....	52
4. Fonction readadc.....	52
Travail Etudiant n°3.....	56
1. Capteur Grove GPS.....	56
2. Récupération, Traitement et stockage des données du GPS.....	58
3. Stockage des données (gaz, température et humidité).....	62
4. Création du site de monitoring et la base de données.....	63
Conclusion.....	74
Annexe.....	75
Etudiant 1.....	75
Etudiant 2.....	80
Etudiant 3.....	82



Introduction

La protection des intervenants sur les chantiers est depuis longtemps devenue une priorité. Que ce soit par le développement de nouvelles techniques ou technologies, par la mise en place de nouvelles normes, l'accidentologie (et donc la mortalité) est en régression depuis plusieurs décennies.

Dernière évolution de la protection individuelle : l'ajout de fonctionnalités «connectées» à un casque de chantier pour améliorer la sécurité de ses utilisateurs. Pour répondre efficacement aux différents besoins, nous intégrerons des solutions adaptées à notre dispositif.

Tous les capteurs seront contrôlés par une carte Raspberry Pi. Ce petit ordinateur embarqué, qui possède toutes les fonctionnalités essentielles d'un ordinateur classique, sera chargé de gérer la communication avec les serveurs.

Pour assurer la sécurité de l'utilisateur dans son environnement, le casque sera équipé de plusieurs capteurs spécifiques. Un capteur de gaz permettra de détecter la présence de substances dangereuses, tandis qu'un capteur d'humidité surveillera la température ambiante. Ces dispositifs auront pour but d'alerter l'utilisateur des potentiels dangers liés au gaz ou à des variations de température.

Pour renforcer la sécurité physique de l'utilisateur, le casque sera équipé d'un accéléromètre. Cet appareil mesurera les coordonnées cartésiennes de l'utilisateur, permettant ainsi de détecter et de confirmer la survenue éventuelle d'une chute.

Pour assurer une visibilité optimale, un capteur de luminosité régulera l'intensité de la lampe frontale en fonction du niveau d'obscurité.

Le casque sera doté d'un capteur de position GPS, permettant au chef de chantier de gérer à distance le suivi des travailleurs. Ce dernier recevra des notifications en temps réel basées sur des données fournies par divers capteurs, tels que ceux détectant le gaz, la température, ou les chutes. De plus, il aura accès à une mini-carte affichant la dernière position connue de chaque utilisateur, ainsi qu'à un historique des notifications pour chaque utilisateur.

Répartition des tâches

Etudiant n°1 : Lilian Bowen

Voici une liste des tâches que doit accomplir l'étudiant n°1 :

1. Coder, tester et valider un module logiciel de détection pour :

- La chute ou l'inconscience du porteur du casque.
- La possibilité d'appeler à l'aide manuellement grâce à un bouton d'alerte.
- Assurer une bonne visibilité de la zone de travail avec un éclairage proportionnel.

2. Coder, tester et valider un module logiciel permettant :

- D'obtenir une interface homme-machine (IHM) simple d'alerte pour le porteur du casque.
- De définir, préparer et transmettre les messages d'alerte au chef de chantier.
- De sauvegarder les données transmises.

Etudiant n°2 : Kyllian Delompré

Voici la liste des tâches que doit accomplir l'étudiant n°2 :

1. Coder, tester et valider un module logiciel de détection pour :

- La température.
- La présence de gaz dangereux.

2. Coder, tester et valider un module logiciel permettant :

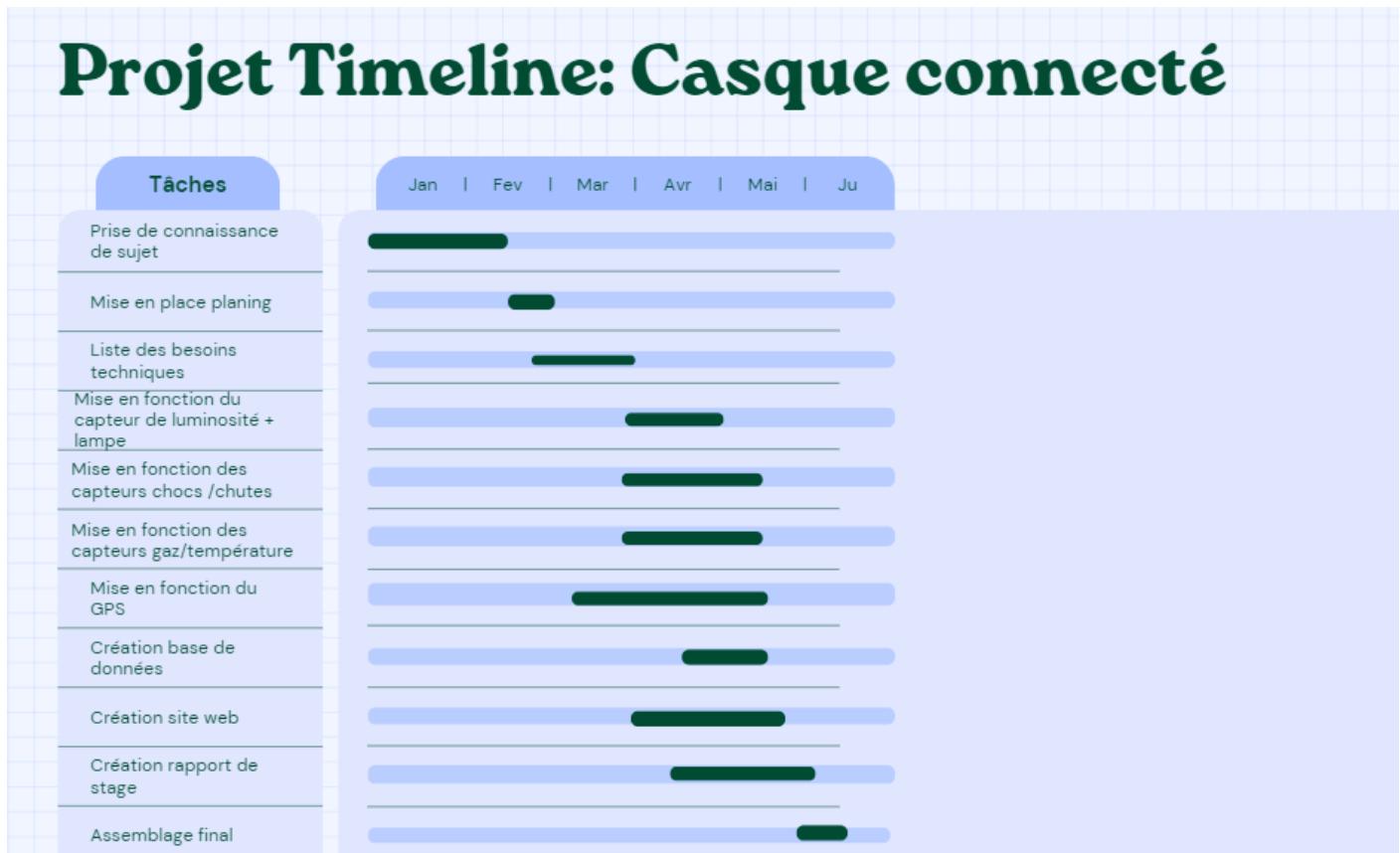
- De déterminer les alertes en fonction des détections effectuées.
- De définir, préparer et transmettre les messages d'alerte au chef de chantier.

- De sauvegarder les données transmises.

Etudiant n°3 : Pedro Gomes

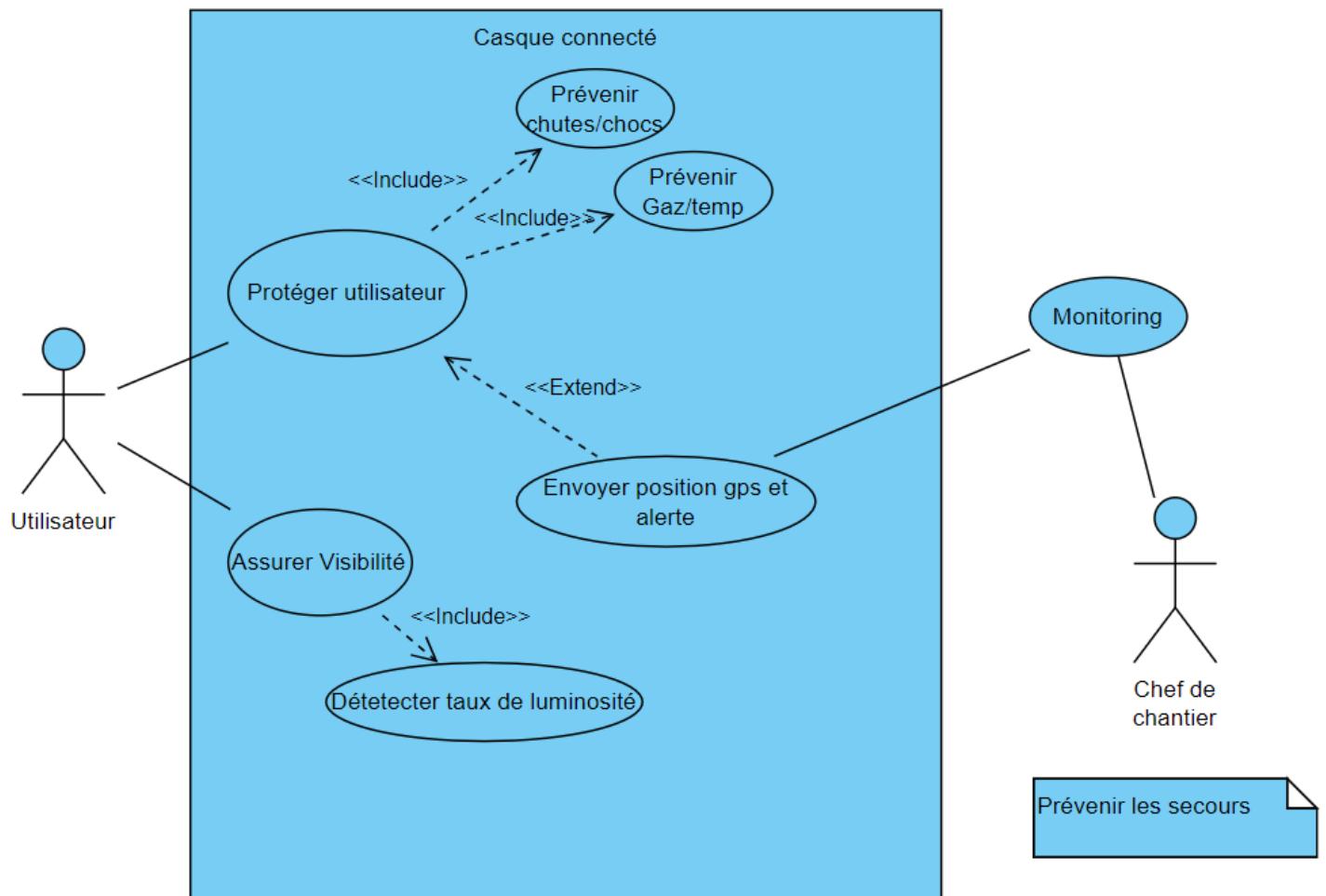
Voici la liste des tâches que doit accomplir l'étudiant n°3 :

1. Coder, tester et valider un module logiciel pour :
 - L'obtention des coordonnées GPS.
2. S'approprier et finaliser la modélisation du système.
3. Installer et configurer le réseau Wifi fermé.
4. Coder, tester et valider un module logiciel permettant :
 - D'obtenir les coordonnées GPS.
 - De décoder les informations reçues du casque de chantier.
 - D'obtenir une base de données centralisant toutes les informations sur le porteur de casque.
 - De créer un site web permettant la gestion des porteurs de casques, incluant des fonctionnalités telles que le suivi des coordonnées GPS, la détection de gaz dangereux et les niveaux de température.

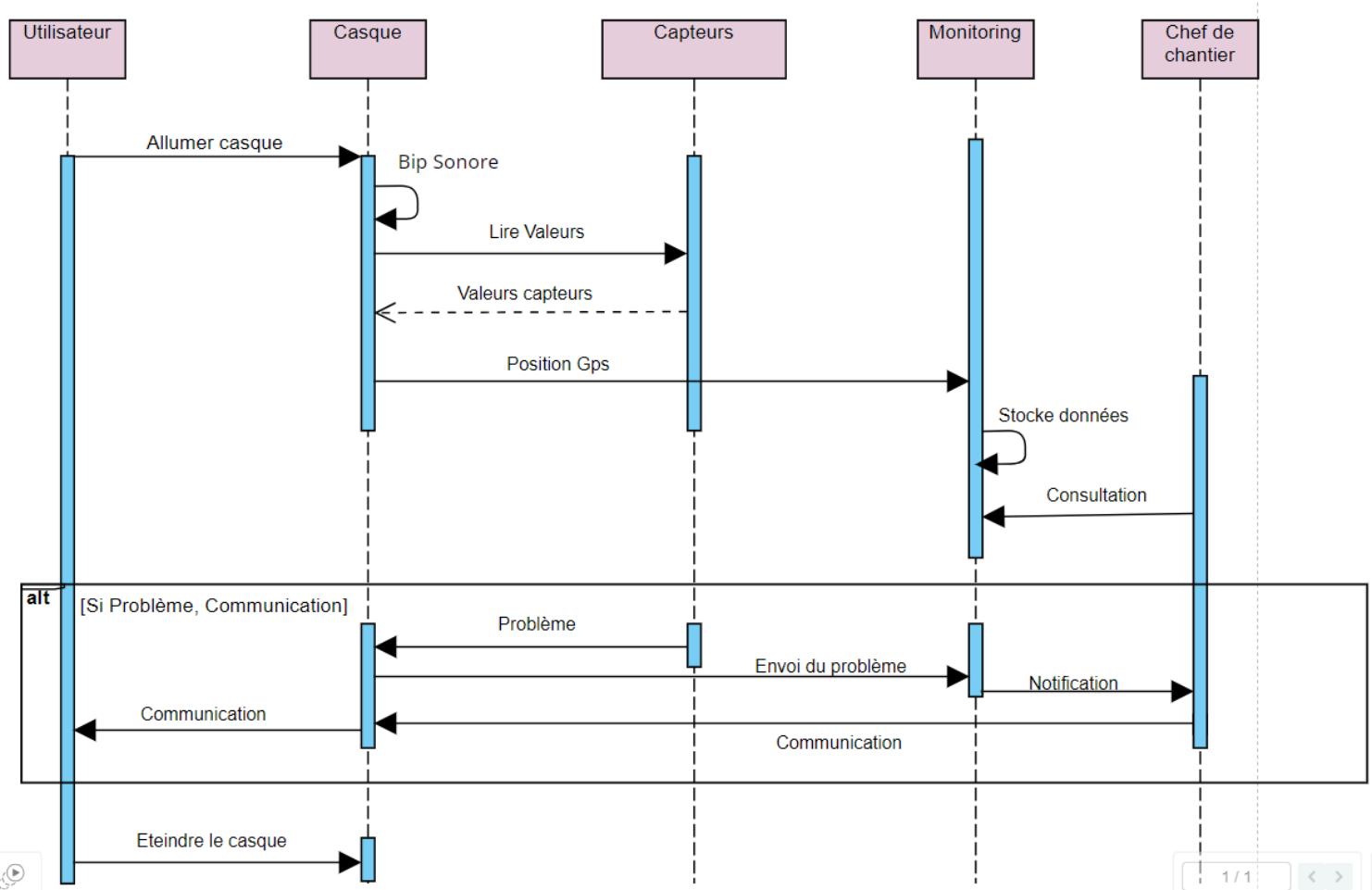


Diagrammes

1. Diagramme de cas d'utilisation

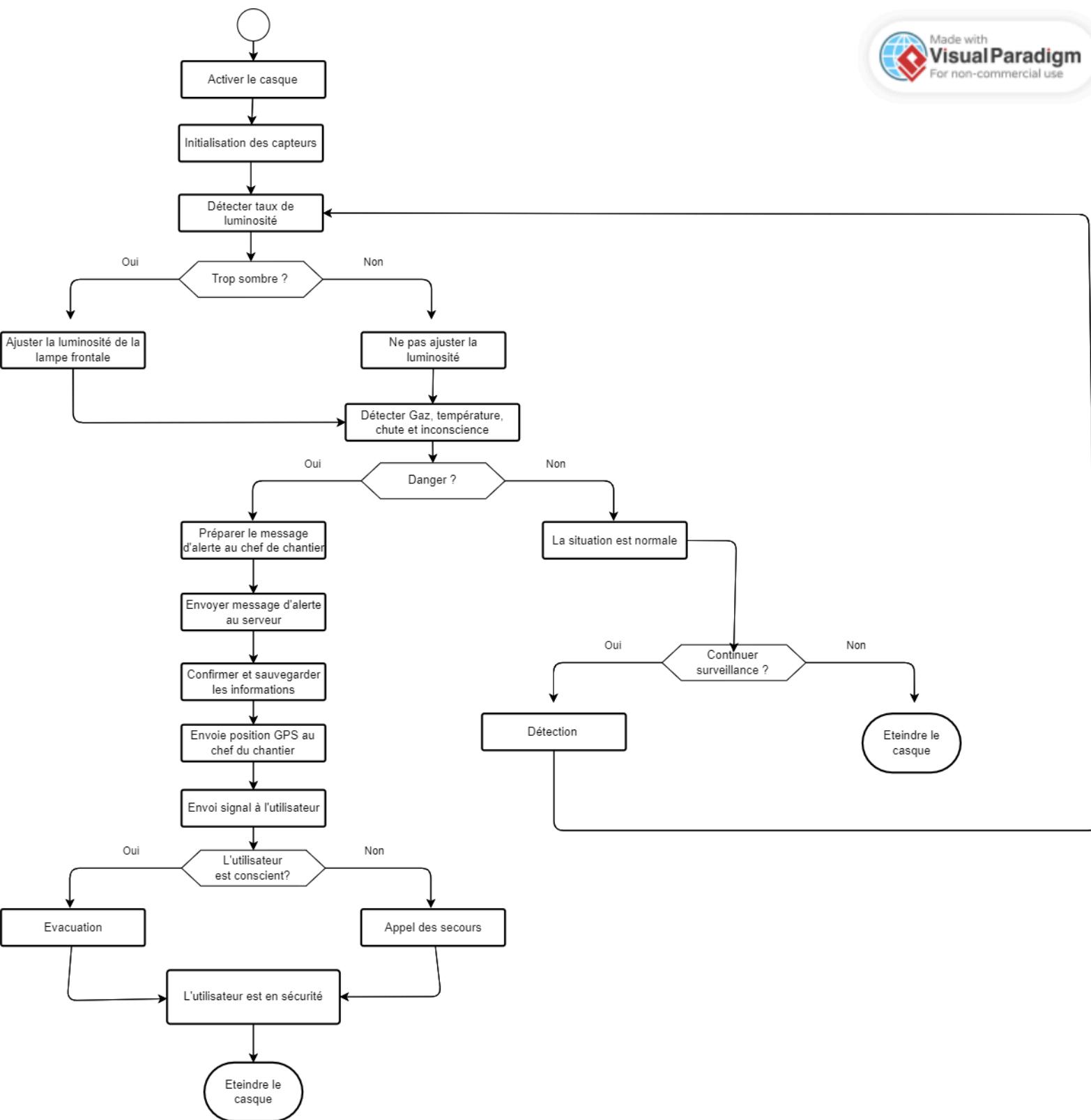


2. Diagramme de séquence



3. Diagramme de Flowchart

Made with
VisualParadigm
For non-commercial use



Travail Etudiant n°1

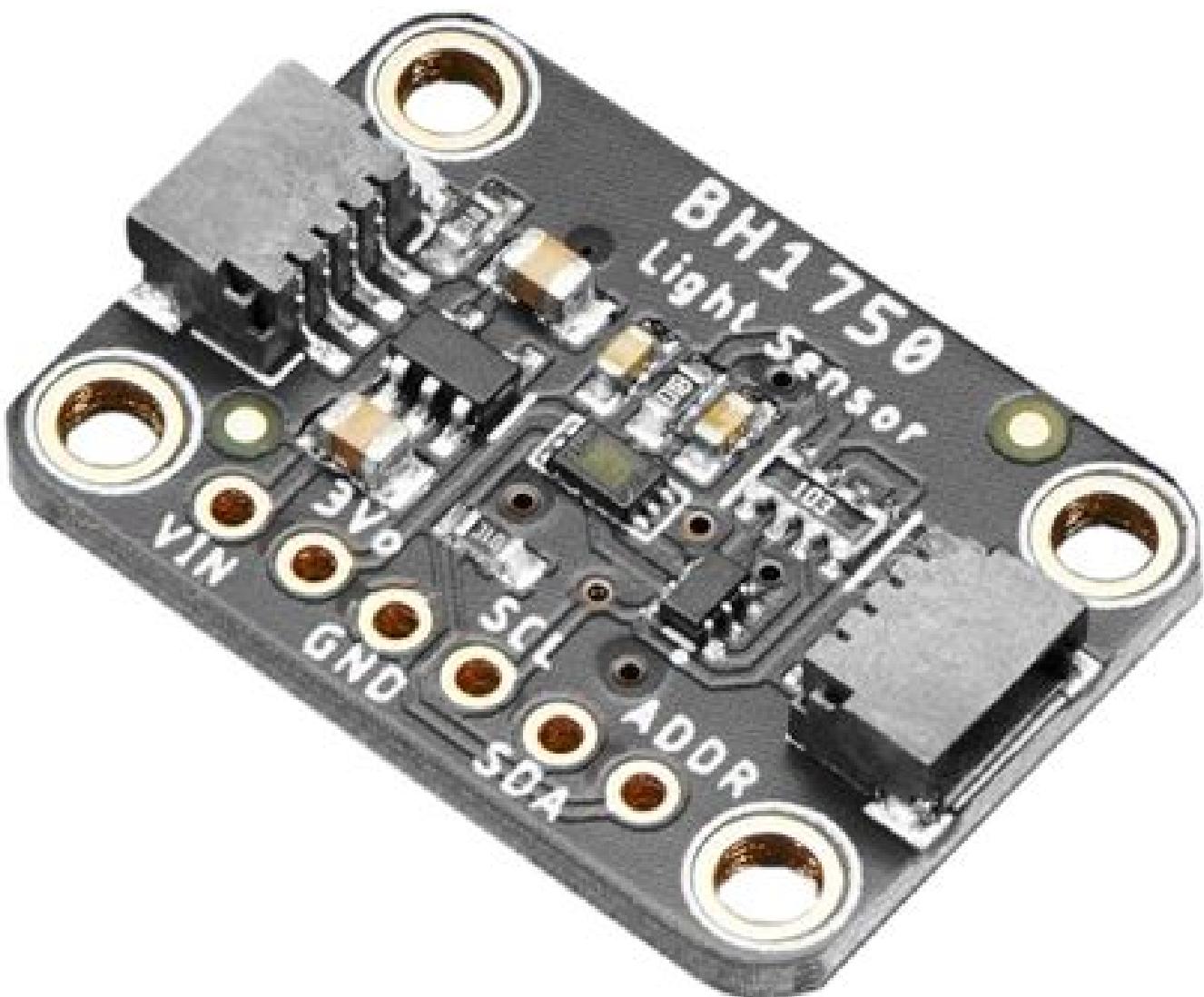
Composants électroniques

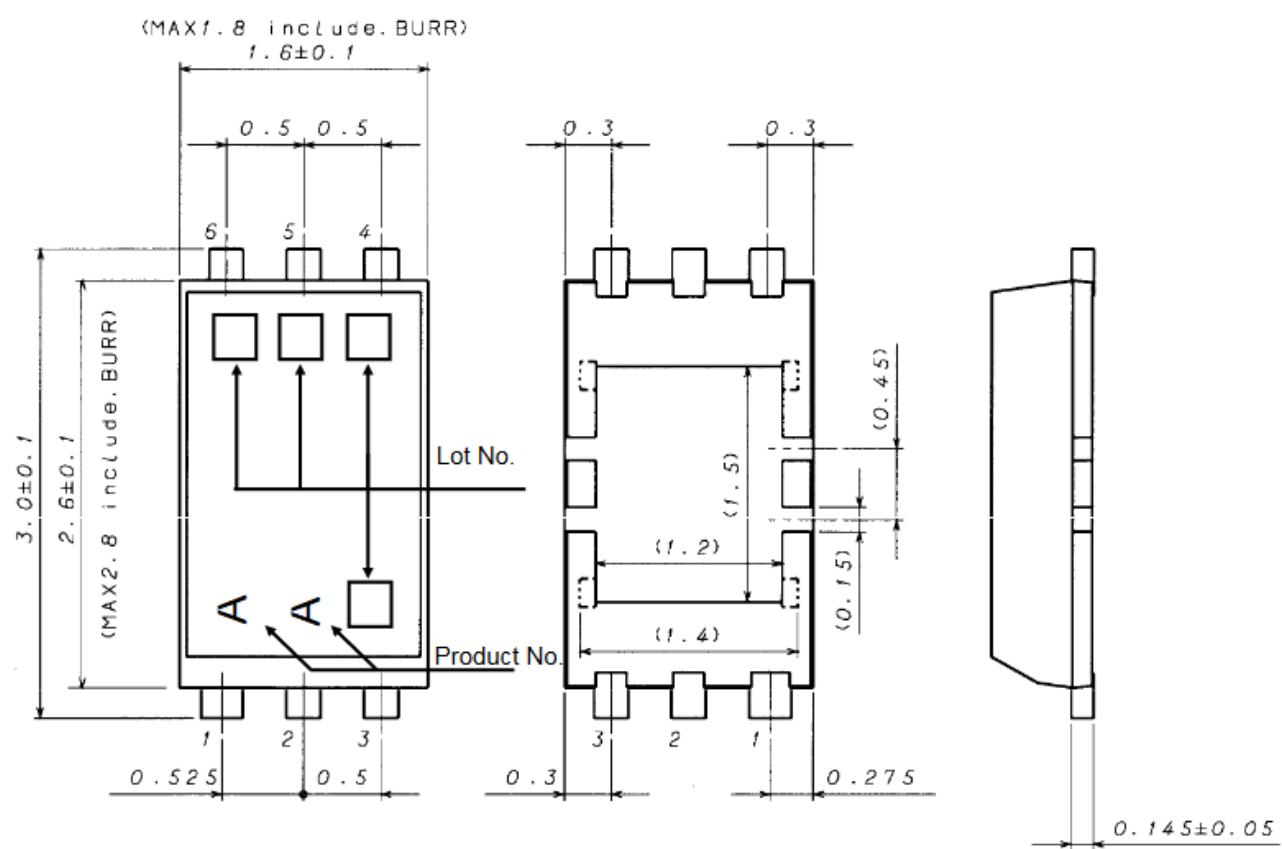
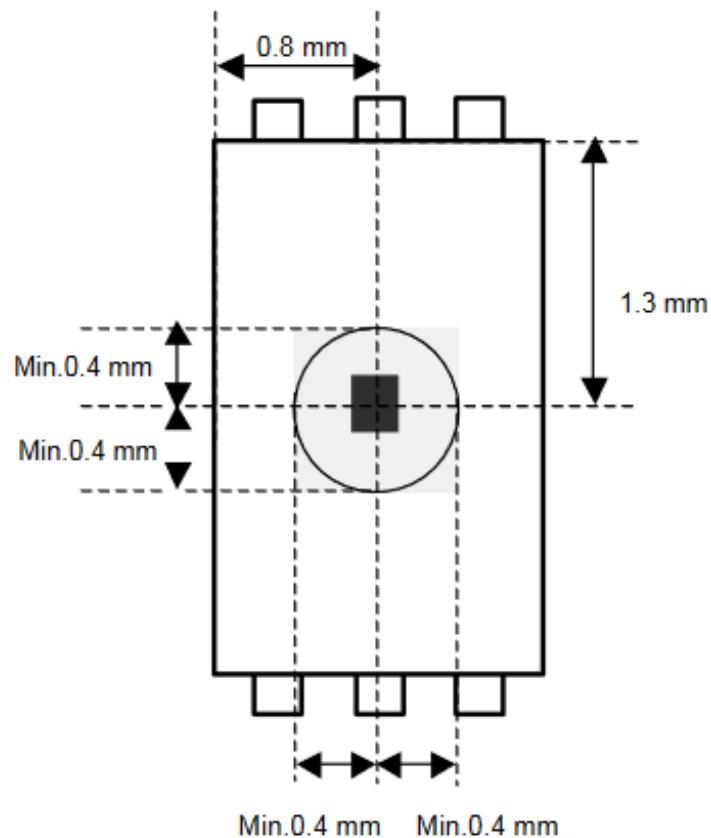
1-Capteur de luminosité:

Le capteur de luminosité BH1750 est un dispositif électronique sophistiqué conçu pour mesurer avec précision l'intensité lumineuse ambiante. Voici une description générale de son fonctionnement :

- Principe de fonctionnement:** Le BH1750 utilise la technologie de la photorésistance, également connue sous le nom de cellule photoélectrique. Cette technologie repose sur la variation de la résistance électrique d'un matériau en fonction de la quantité de lumière qu'il reçoit.
- Composants:** Le capteur BH1750 est équipé d'un photorécepteur sensible à la lumière, qui convertit la lumière incidente en un signal électrique mesurable. Ce signal est ensuite traité par un circuit électronique intégré pour fournir une lecture numérique de l'intensité lumineuse.
- Communication numérique:** Le capteur BH1750 est capable de communiquer avec d'autres dispositifs électroniques via des interfaces numériques telles que I²C (Inter-Integrated Circuit) ou SPI (Serial Peripheral Interface), ce qui le rend facilement intégrable dans différents systèmes.
- Calibrage automatique:** Le BH1750 est souvent équipé d'un circuit de calibrage automatique qui ajuste sa sensibilité en fonction des conditions d'éclairage ambiantes, assurant ainsi des mesures précises dans une large gamme de conditions lumineuses.
- Haute résolution et précision:** Ce capteur est apprécié pour sa haute résolution et sa précision, ce qui le rend idéal pour des applications telles que la domotique, la gestion de l'éclairage, la météorologie, et d'autres systèmes nécessitant une surveillance précise de la luminosité.

En résumé, le BH1750 est un capteur de luminosité fiable et précis, largement utilisé dans diverses applications pour surveiller et contrôler l'intensité lumineuse ambiante de manière efficace et précise.



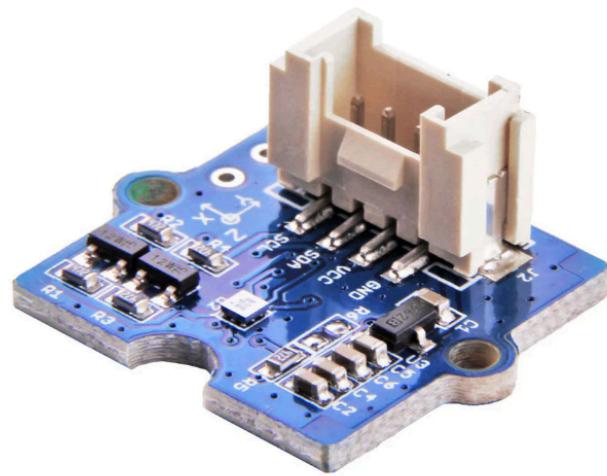


2. Accéléromètre :

Un accéléromètre 3 axes digital est un capteur électronique utilisé pour mesurer l'accélération dans trois directions spatiales différentes : x, y et z. Voici une description générale de son fonctionnement :

1. Principe de fonctionnement: L'accéléromètre 3 axes digital repose généralement sur le principe de la force exercée par l'accélération sur une masse interne, généralement un élément piézoélectrique ou capacitif. Lorsque l'appareil subit une accélération, cette masse interne est déplacée, ce qui génère un signal électrique proportionnel à l'accélération subie.
2. Axes de mesure: Un accéléromètre 3 axes mesure l'accélération dans trois directions perpendiculaires les unes aux autres, généralement désignées comme x, y et z. Ces axes sont définis en fonction de l'orientation de l'accéléromètre dans l'espace.
3. Capteurs internes: À l'intérieur de l'accéléromètre, il y a généralement trois capteurs distincts, un pour chaque axe de mesure. Chaque capteur est sensible à l'accélération le long de son axe spécifique.
4. Conversion analogique-numérique: Les signaux électriques générés par les capteurs internes sont ensuite convertis en signaux numériques par un convertisseur analogique-numérique (CAN), ce qui permet leur traitement par un microcontrôleur ou un processeur.
5. Filtrage et calibrage: Les données mesurées peuvent être filtrées pour éliminer les fluctuations indésirables ou les bruits, et l'accéléromètre peut être calibré pour compenser les dérives ou les erreurs de mesure.
6. Interfaces de communication: Les accéléromètres 3 axes digital peuvent généralement communiquer avec d'autres dispositifs électroniques via des interfaces telles que l'I²C (Inter-Integrated Circuit) ou le SPI (Serial Peripheral Interface), ce qui permet leur intégration dans divers systèmes électroniques.
7. Applications: Les accéléromètres 3 axes sont largement utilisés dans une variété d'applications, y compris la navigation inertielle, la détection des mouvements dans les dispositifs portables, la stabilisation d'image dans les caméras, les jeux vidéo, la surveillance de l'activité physique, et bien plus encore.

En résumé, un accéléromètre 3 axes digital est un capteur essentiel pour mesurer et quantifier l'accélération dans trois dimensions, ce qui le rend précieux dans de nombreuses applications où la détection des mouvements ou la mesure de l'inclinaison est nécessaire.



ADXL345
TOP VIEW
(Not to Scale)

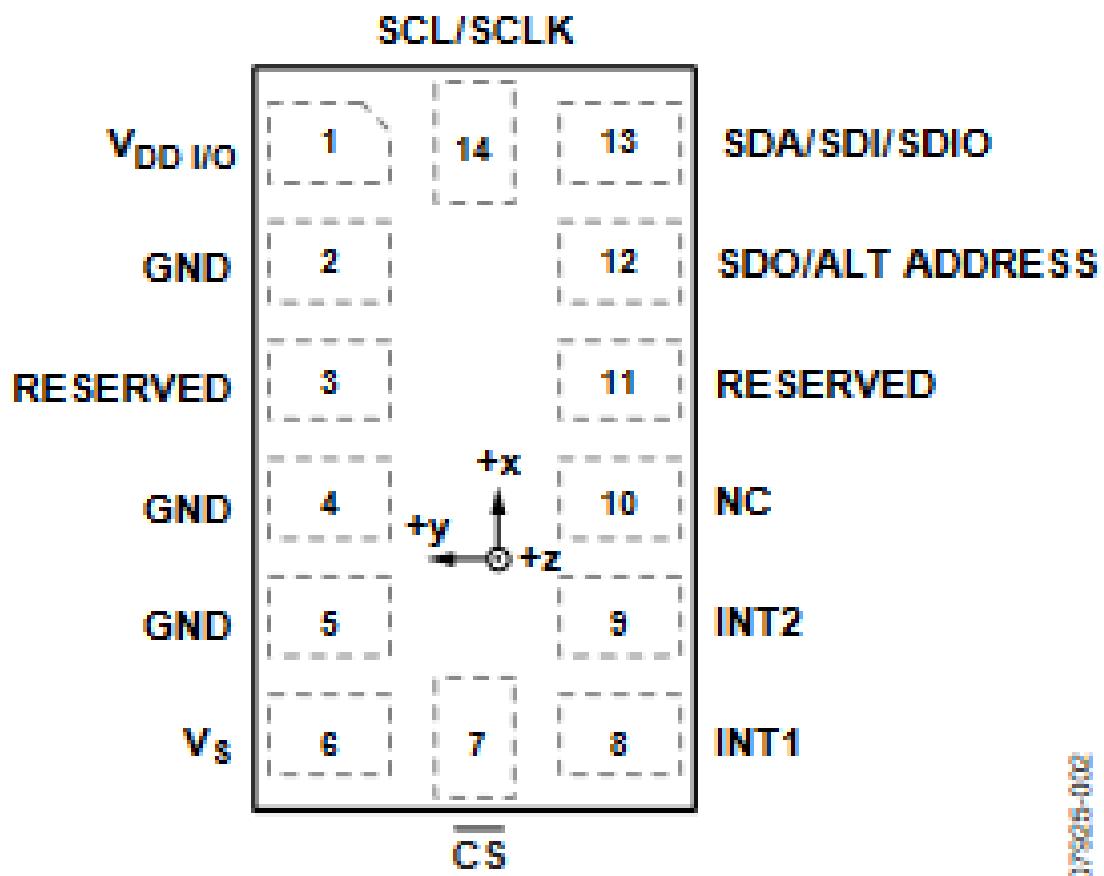


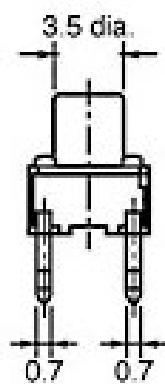
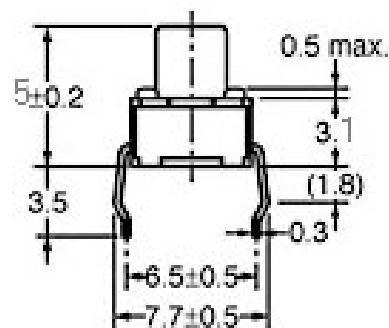
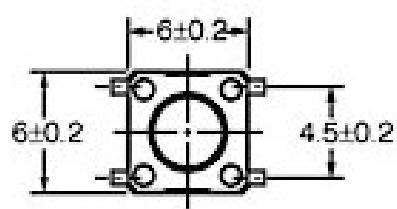
Figure 3. Pin Configuration (Top View)

3. Bouton poussoir :

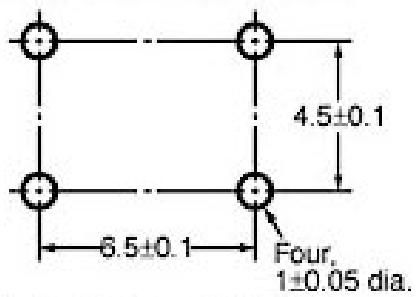
Un bouton poussoir est un composant électromécanique simple utilisé pour établir ou interrompre un circuit électrique. Voici une description générale de son fonctionnement :

1. Structure physique: Un bouton poussoir est généralement composé d'un corps en plastique ou en métal contenant un contact électrique, un ressort et un bouton. Le bouton est enfoncé pour établir le contact électrique et relâché pour ouvrir le circuit.
2. Contact électrique: À l'intérieur du bouton poussoir, il y a au moins deux contacts électriques. Lorsque le bouton est enfoncé, ces contacts se touchent, permettant ainsi le passage du courant électrique à travers le circuit.
3. Actionnement mécanique: Lorsque l'utilisateur exerce une pression sur le bouton, le mécanisme de ressort comprime, poussant les contacts électriques l'un contre l'autre pour fermer le circuit. Lorsque la pression est relâchée, le ressort ramène les contacts à leur position initiale, ouvrant ainsi le circuit.
4. Utilisation en circuit: Les boutons poussoirs peuvent être utilisés de différentes manières dans les circuits électriques. Ils peuvent être configurés en tant que boutons "normalement ouverts" (NO) ou "normalement fermés" (NF), selon qu'ils établissent ou interrompent le circuit par défaut.
5. Applications: Les boutons poussoirs sont largement utilisés dans de nombreux appareils électroniques et systèmes de contrôle. Ils peuvent être trouvés dans les télécommandes, les claviers d'ordinateurs, les dispositifs d'interface utilisateur, les circuits d'éclairage, et bien d'autres applications.
6. Fiabilité et durabilité: Les boutons poussoirs sont généralement conçus pour être robustes et fiables, capables de supporter un grand nombre de cycles d'activation sans défaillance.

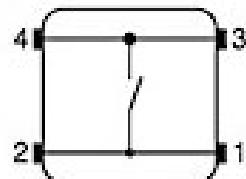
En résumé, un bouton poussoir est un composant simple mais essentiel dans de nombreux appareils électroniques, offrant une interface utilisateur pratique pour établir ou interrompre des circuits électriques en appuyant simplement sur un bouton.



PCB Mounting (Top View)
(Single-sided PCB, t=1.6)



Terminal Arrangement / Internal Connections (Top View)



Programmes Python

1. Programmation Bouton poussoir:

```
import RPi.GPIO as GPIO
from time import sleep, time

# Configuration des GPIO
BUTTON_PIN = 17 # Numéro de la broche GPIO pour le bouton
RESPONSE_TIMEOUT = 10 # Délai d'attente pour la réponse en secondes

GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def check_button_response():
    print("Est-ce que tout va bien ? Appuyez sur le bouton pour confirmer.")

    start_time = time()
    button_pressed = False

    while time() - start_time < RESPONSE_TIMEOUT:
        if GPIO.input(BUTTON_PIN) == GPIO.LOW: # Le bouton est appuyé
            button_pressed = True
            break
        sleep(0.1) # Pause de 100ms pour éviter de vérifier trop fréquemment

    if button_pressed:
        print("Bouton appuyé, tout va bien.")
    else:
        print("Alerte : Pas de réponse du bouton !")

try:
    while True:
        check_button_response()
        sleep(30) # Vérifie toutes les 30 secondes

except KeyboardInterrupt:
    print("Programme arrêté par l'utilisateur")

finally:
    GPIO.cleanup()
```

Explications :

1. Configuration des GPIO :

- Utilise la broche GPIO 17 pour le bouton poussoir.
- Configure le bouton poussoir avec une résistance de pull-up interne.

1. Fonction `check_button_response` :

- Affiche un message demandant si tout va bien.
- Attend que le bouton soit pressé ou que le délai expire.
- Si le bouton est pressé, affiche un message confirmant que tout va bien.
- Si le délai expire sans réponse, affiche un message d'alerte.

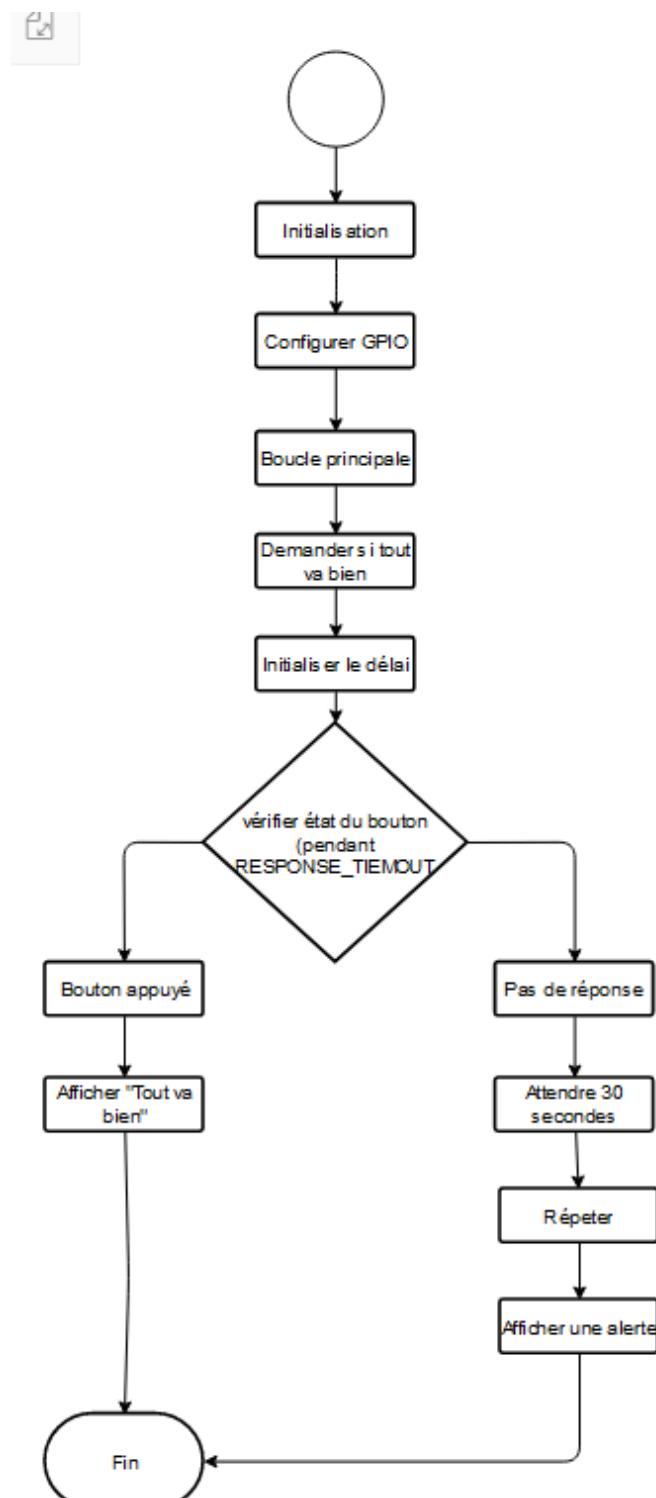
2. Boucle Principale :

- Appelle la fonction `check_button_response` toutes les 30 secondes.
- Interrupt proprement le programme si l'utilisateur appuie sur Ctrl+C.

3. Nettoyage :

- Nettoie les configurations GPIO à la fin du programme.

Ce code fournit une base simple pour interroger l'état d'un bouton poussoir et gérer une alerte en cas de non-réponse. Vous pouvez adapter la partie d'alerte pour envoyer un message, déclencher une alarme, ou toute autre action appropriée à votre application.



2. Programmation Accéléromètre :

1. Importation des bibliothèques :

```
import smbus  
from time import sleep
```

Ces instructions importent les bibliothèques nécessaires. smbus est utilisé pour la communication I2C avec l'accéléromètre, et sleep est utilisé pour introduire des délais dans le programme.

Définition des adresses et constantes :

```
EARTH_GRAVITY_MS2 = 9.80665  
SCALE_MULTIPLIER = 0.004
```

Ces constantes sont utilisées pour convertir les valeurs brutes de l'accéléromètre en unités d'accélération (m/s^2).

Initialisation du bus I2C :

```
revision = ([l[12:-1] for l in open('/proc/cpuinfo','r').readlines() if l[:8]== "Revision"]+[0])  
bus = smbus.SMBus(1 if int(revision, 16) >= 4 else 0)
```

Cette partie du code détermine la révision du Raspberry Pi pour sélectionner le bon bus I2C. Ensuite, elle initialise le bus I2C.

Définition des constantes pour l'accéléromètre :

```
DATA_FORMAT = 0x31  
BW_RATE = 0x2C  
POWER_CTL = 0x2D
```

Ces constantes représentent les adresses des registres de configuration de l'accéléromètre ADXL345.

Définition de la classe ADXL345 :

Cette partie du code définit une classe ADXL345 pour interagir avec l'accéléromètre. Les méthodes de cette classe permettent de configurer l'accéléromètre et de récupérer les valeurs des axes x, y et z.

Boucle principale :

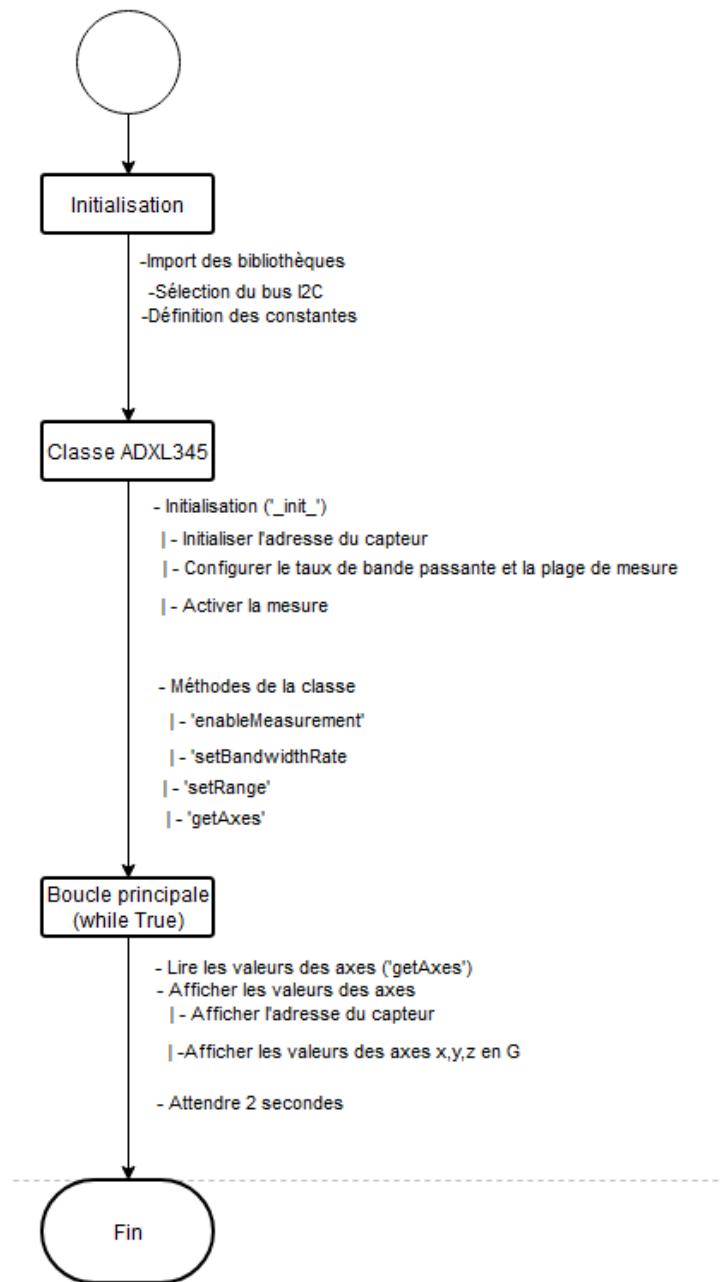
```
if __name__ == "__main__":
    adxl345 = ADXL345()

    while True:
        axes = adxl345.getAxes(True)
        print ("ADXL345 on address 0x%x:" % (adxl345.address))
        print ("    x = %.3fG" % ( axes['x'] ))
        print ("    y = %.3fG" % ( axes['y'] ))
        print ("    z = %.3fG" % ( axes['z'] ))

        # Attendre une seconde avant de récupérer et d'afficher de nouvelles données
        sleep(2)
```

Dans la boucle principale, une instance de la classe ADXL345 est créée, puis les valeurs des axes x, y et z de l'accéléromètre sont lues et affichées en continu. Une pause de deux secondes est introduite entre chaque lecture pour éviter une surcharge de données.

Cette structure permet de lire en continu les valeurs d'accélération de l'accéléromètre et de les afficher à l'écran.



3. Programmation Capteur de lumière

Partie 1 : Importation des bibliothèques

```
import smbus  
import time  
import RPi.GPIO as GPIO
```

Explication :

- smbus : Permet la communication I2C avec des périphériques comme le capteur de luminosité BH1750.
- time : Utilisé pour les temporisations dans le programme.
- RPi.GPIO : Permet de contrôler les GPIO du Raspberry Pi.

Partie 2 : Définition des adresses et modes

```
# Définition des adresses des capteur BH1750  
BH1750_ADDR = 0x23  
# Mode de mesure continu de 1 lux  
CONTINUOUS_HIGH_RES_MODE = 0x10
```

Explication :

- BH1750_ADDR : Adresse I2C du capteur de luminosité BH1750.
- CONTINUOUS_HIGH_RES_MODE : Mode de mesure du capteur qui fournit une résolution élevée et continue des mesures de luminosité.

Partie 3 : Initialisation du bus I2C

```
# Initialisation du bus I2C  
bus = smbus.SMBus(1)
```

Explication :

- Initialise le bus I2C pour permettre la communication avec le capteur de luminosité. 1 signifie que nous utilisons le bus I2C-1.

Partie 4 : Configuration des pins GPIO

```
# Configuration des pins GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(16, GPIO.OUT)
```

Explication :

- GPIO.setmode(GPIO.BOARD) : Utilise le numéro de pin physique sur le connecteur du GPIO.
- GPIO.setup(16, GPIO.OUT) : Configure le pin 16 comme une sortie, probablement pour contrôler une LED.

Partie 5 : Initialisation du PWM

```
# initialisation de pwm_led à None
pwm_led = None
```

Explication :

- read_light : Fonction qui lit la luminosité du capteur.
 - data : Lit 2 octets de données du capteur dans le mode haute résolution continue.
 - light_level : Convertit les données de luminosité en un entier.

Partie 7 : Boucle principale

```
try:
    while True:
        light_level = read_light()
        brightness = max(100 - int(light_level / 2), 0) # Inverser le calcul de la lumi

        # Vérifier si un objet PWM existe déjà
        if pwm_led:
            pwm_led.stop() # Arrêter l'objet PWM existant
            pwm_led = None # Réinitialiser l'objet PWM existant

        # Créer un nouvel objet PWM
        pwm_led = GPIO.PWM(16, 100)
        pwm_led.start(brightness) # Démarrer avec la nouvelle luminosité

        time.sleep(1)
except KeyboardInterrupt:
    print("Arrêt du programme par l'utilisateur")
finally:
    if pwm_led:
        pwm_led.stop() # Arrêter l'objet PWM s'il existe encore
    GPIO.cleanup()
```

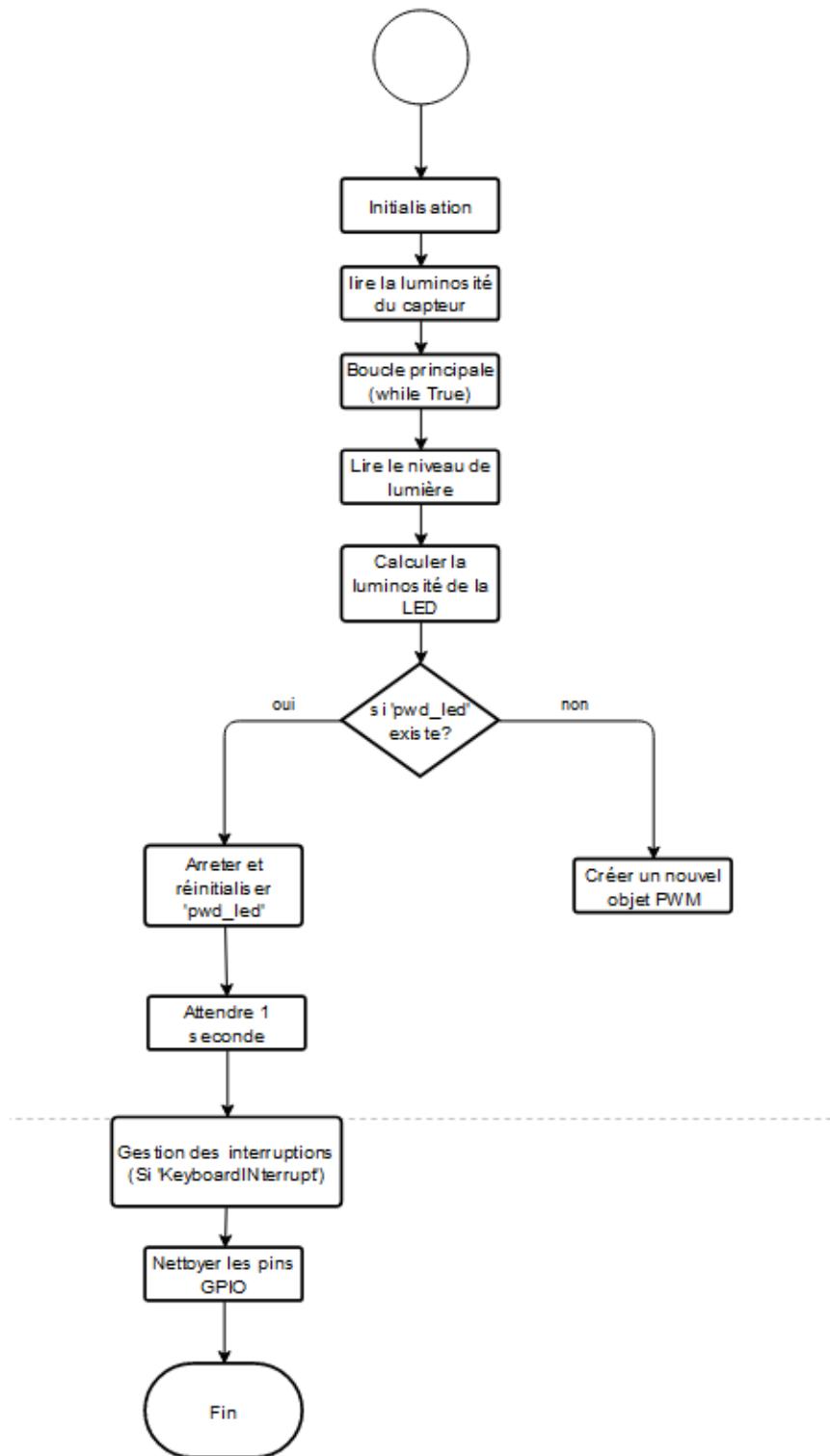
Explication :

- **Boucle principale (try):**
 - **Lecture de la luminosité** : Appelle la fonction `read_light` pour obtenir le niveau de luminosité.
 - **Calcul de la luminosité de la LED** : Calcule la luminosité de la LED en inversant la valeur de la luminosité lue (plus il fait sombre, plus la LED est lumineuse).
- **Contrôle PWM :**
 - Vérifie si un objet PWM existe déjà et le stoppe si nécessaire.
 - Crée un nouvel objet PWM pour contrôler la LED avec la nouvelle valeur de luminosité.
- **Temporisation** : Attend 1 seconde avant de refaire une lecture.

- **Gestion des exceptions (except KeyboardInterrupt):**
 - Arrête le programme proprement si l'utilisateur appuie sur Ctrl+C.
- **Nettoyage final (finally):**
 - Arrête l'objet PWM s'il existe.
 - Nettoie les configurations GPIO pour libérer les ressources.

Résumé

Ce code lit les valeurs de luminosité du capteur BH1750 et ajuste la luminosité d'une LED en fonction de ces valeurs. Plus il fait sombre, plus la LED est lumineuse. Le bouton poussoir et d'autres composants peuvent être ajoutés selon les besoins de l'application.



Résumé :

Ce code configure un capteur de luminosité BH1750 pour mesurer la luminosité ambiante via le bus I2C sur une Raspberry Pi. Il utilise les lectures de luminosité pour ajuster la luminosité d'une LED connectée à la broche GPIO 16. La luminosité de la LED est contrôlée par PWM, où la LED devient plus brillante dans l'obscurité et plus faible à la lumière. Le code gère également les interruptions utilisateur (Ctrl+C) pour arrêter proprement le programme et réinitialiser les broches GPIO.

4 Bouton poussoir + Accéléromètre:

1. Importation des bibliothèques et des constantes :

```
import smbus
from time import sleep, time
import RPi.GPIO as GPIO

BUTTON_PIN = 17 # Numéro de la broche GPIO pour le bouton
RESPONSE_TIMEOUT = 10 # Délai d'attente pour la réponse en secondes
```

- Nous importons les bibliothèques nécessaires pour la communication avec l'accéléromètre, la gestion des délais et des GPIO.
- Les constantes définissent le numéro de broche du bouton et le délai d'attente pour la réponse du bouton.

2. Initialisation de l'accéléromètre :

```
bus = smbus.SMBus(1)
adxl345_address = 0x53 # Adresse de l'accéléromètre
adxl345 = ADXL345(adxl345_address)
```

- Nous initialisons le bus I2C pour communiquer avec l'accéléromètre.
- Nous créons une instance de la classe ADXL345 avec l'adresse spécifiée de l'accéléromètre.

3. Fonction check_button_response :

```
def check_button_response():
    print("Est-ce que tout va bien ? Appuyez sur le bouton pour confirmer.")

    start_time = time()
    button_pressed = False

    while time() - start_time < RESPONSE_TIMEOUT:
        if GPIO.input(BUTTON_PIN) == GPIO.LOW: # Le bouton est appuyé
            button_pressed = True
            break
        sleep(0.1) # Pause de 100ms pour éviter de vérifier trop fréquemment

    if button_pressed:
        print("Bouton appuyé, tout va bien.")
    else:
        print("Alerte : Pas de réponse du bouton !")
```

Cette fonction demande à l'utilisateur s'il va bien et attend une réponse du bouton pendant un certain délai () .

Si le bouton est enfoncé pendant ce délai, elle affiche un message indiquant que tout va bien. Sinon, elle affiche un message d'alerte.

4. Fonction :

```
def check_acceleration():
    axes = adxl345.getAxes(True)
    x_acceleration = axes['x']
    y_acceleration = axes['y']
    z_acceleration = axes['z']

    if (x_acceleration < ACCEPTABLE_RANGE_MIN or x_acceleration > ACCEPTABLE_RANGE_MAX or
        y_acceleration < ACCEPTABLE_RANGE_MIN or y_acceleration > ACCEPTABLE_RANGE_MAX or
        z_acceleration < ACCEPTABLE_RANGE_MIN or z_acceleration > ACCEPTABLE_RANGE_MAX):
        print("Alerte : Problème avec les coordonnées !")
```

- Cette fonction récupère les valeurs d'accélération des axes x, y et z à partir de l'accéléromètre.
- Elle vérifie si ces valeurs se situent dans la plage acceptable définie par les constantes ACCEPTABLE_RANGE_MIN et ACCEPTABLE_RANGE_MAX.
- Si une valeur d'accélération est en dehors de la plage acceptable, elle affiche un message d'alerte.

5. Boucle principale :

```
try:
    while True:
        check_button_response()
        check_acceleration()
        sleep(30) # Vérifie toutes les 30 secondes

except KeyboardInterrupt:
    print("Programme arrêté par l'utilisateur")

finally:
    GPIO.cleanup()
```

- La boucle principale exécute en continu les fonctions de vérification de la réponse du bouton et des valeurs d'accélération.
- Elle capture une interruption du clavier (KeyboardInterrupt) pour arrêter le programme proprement.

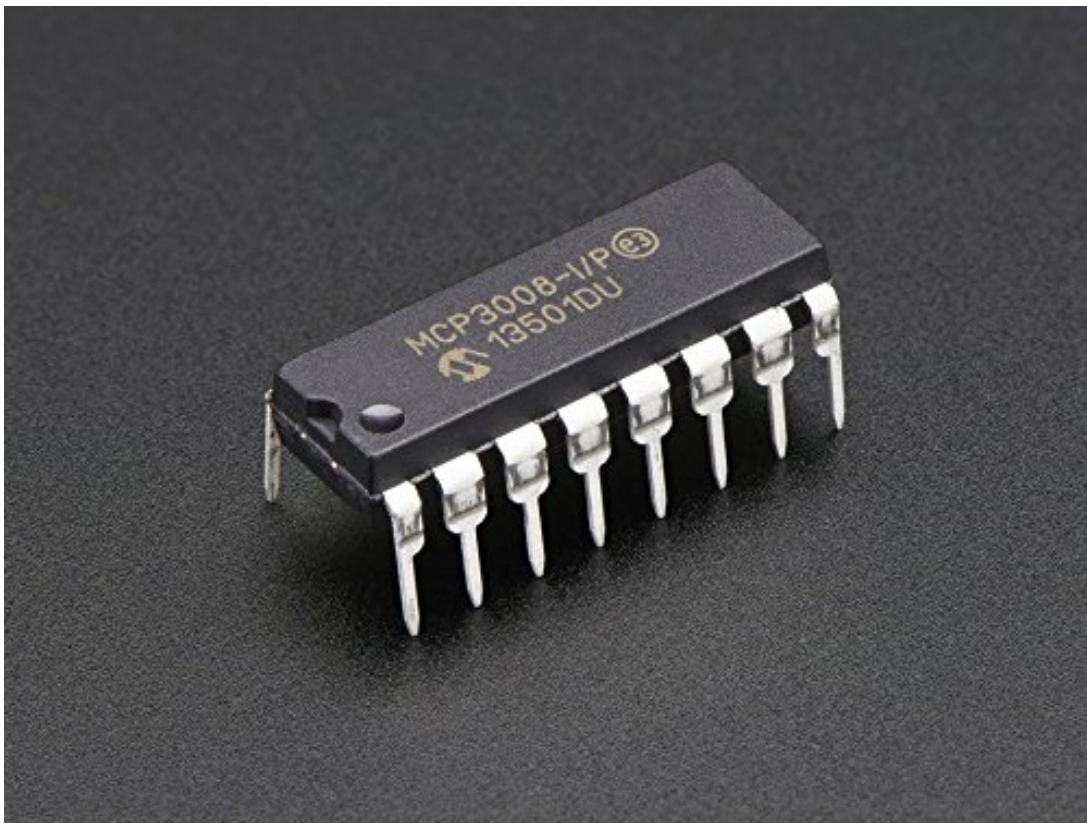
- Enfin, elle nettoie les configurations GPIO pour éviter les problèmes potentiels lors de l'exécution ultérieure du programme.

Ce code combine la fonctionnalité de vérification de la réponse du bouton avec celle de surveillance des valeurs d'accélération de l'accéléromètre. Si un problème est détecté avec les coordonnées (c'est-à-dire si les valeurs d'accélération sont en dehors de la plage acceptable), le programme affiche un message d'alerte.

Travail Etudiant n°2

1. Convertisseur analogique

Au début de notre projet nous utilisions un Shield pour la carte Raspberry Grove Pi+. Malheureusement nous avons rencontré beaucoup de problèmes avec celui-ci, nous avons donc optés un convertisseur analogique MCP3008 :



Le MCP3008 est un convertisseur analogique/digital (ADC) de la série Microchip Technology. C'est un circuit intégré (IC) qui convertit des signaux analogiques en valeurs numériques, ce qui le rend utile dans une variété de projets électroniques, en particulier pour interfaçer des capteurs analogiques avec des microcontrôleurs numériques comme le Raspberry Pi ou Arduino.

Description Générale

Le MCP3008 est un convertisseur analogique/digital à 10 bits offrant huit canaux d'entrée analogiques. Il fonctionne via un protocole de communication SPI (Serial Peripheral Interface), ce qui le rend compatible avec de nombreux microcontrôleurs et plateformes embarquées. Les caractéristiques principales du MCP3008 incluent :

- Résolution : 10 bits, ce qui signifie qu'il peut représenter des valeurs numériques de 0 à 1023.
- Nombre de Canaux : 8 entrées analogiques.
- Type de Communication : SPI, un protocole de communication série synchrone.
- Tension d'Alimentation : De 2,7V à 5,5V, ce qui le rend adaptable à de nombreux environnements.
- Fréquence d'Échantillonnage : Jusqu'à 75 ksps (échantillons par seconde).

Fonctionnement

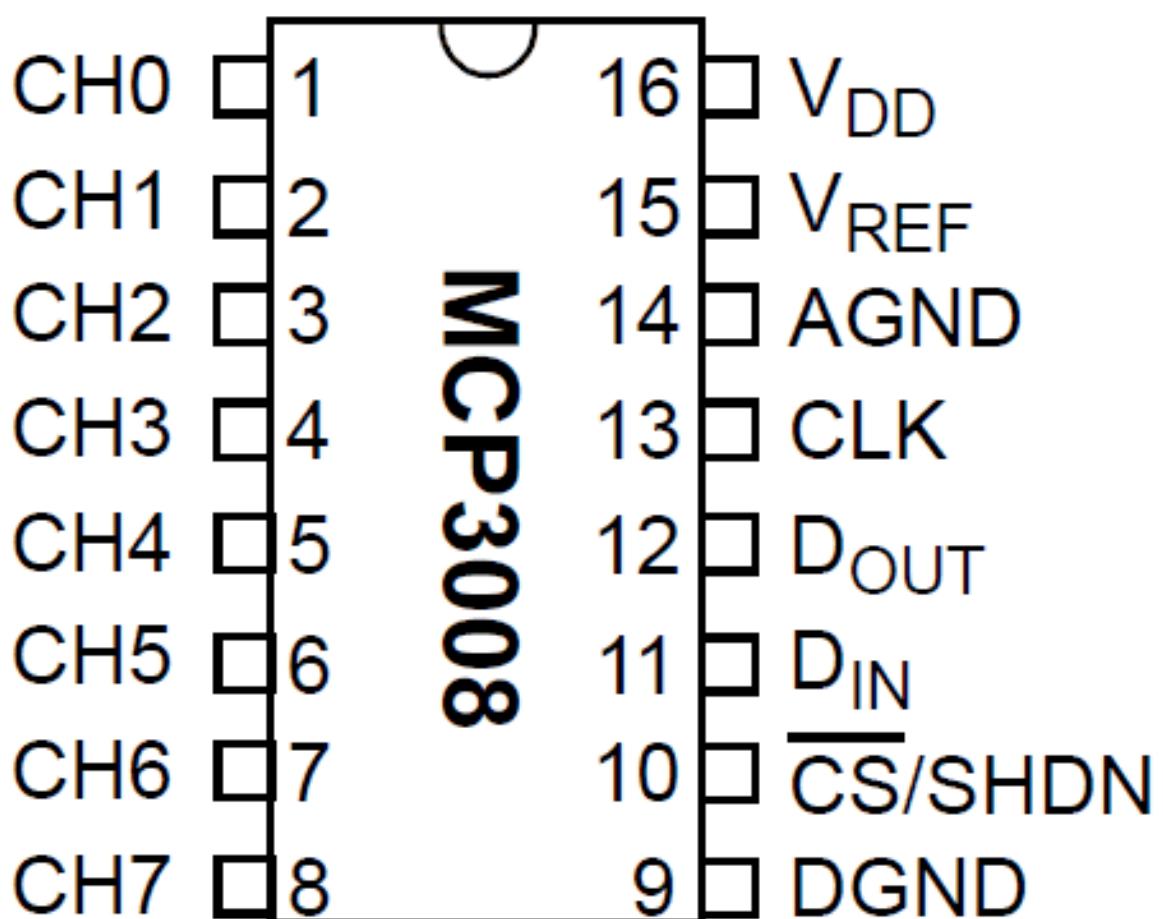
Le MCP3008 fonctionne en convertissant un signal analogique en une valeur numérique. Voici un aperçu de son fonctionnement :

1. Entrées Analogiques : Le MCP3008 dispose de huit entrées analogiques, qui peuvent être configurées comme huit canaux indépendants ou quatre paires différentielles.
2. Conversion A/D : Lorsqu'un signal analogique est appliqué à l'une des entrées, le MCP3008 convertit ce signal en une valeur numérique en utilisant une technique d'approximation successive (SAR, Successive Approximation Register). La résolution de 10 bits permet des valeurs numériques comprises entre 0 et 1023.
3. Communication SPI : Le MCP3008 communique avec le microcontrôleur ou le micro-ordinateur via le protocole SPI. Il nécessite quatre fils pour fonctionner :
 - CS/SS (Chip Select/Slave Select) : Pour sélectionner le MCP3008 pour la communication.

- MOSI (Master Out, Slave In) : Pour envoyer des commandes du microcontrôleur vers le MCP3008.
 - MISO (Master In, Slave Out) : Pour recevoir les données converties du MCP3008.
 - SCK (Serial Clock) : Pour synchroniser les transferts de données.
4. Lecture des Données : Le microcontrôleur envoie une commande via SPI pour lire une entrée spécifique. Le MCP3008 retourne ensuite la valeur numérique correspondante à cette entrée.

Schéma du Fonctionnement

Voici un schéma simplifié montrant comment le MCP3008 fonctionne avec un microcontrôleur comme le Raspberry Pi :



Applications

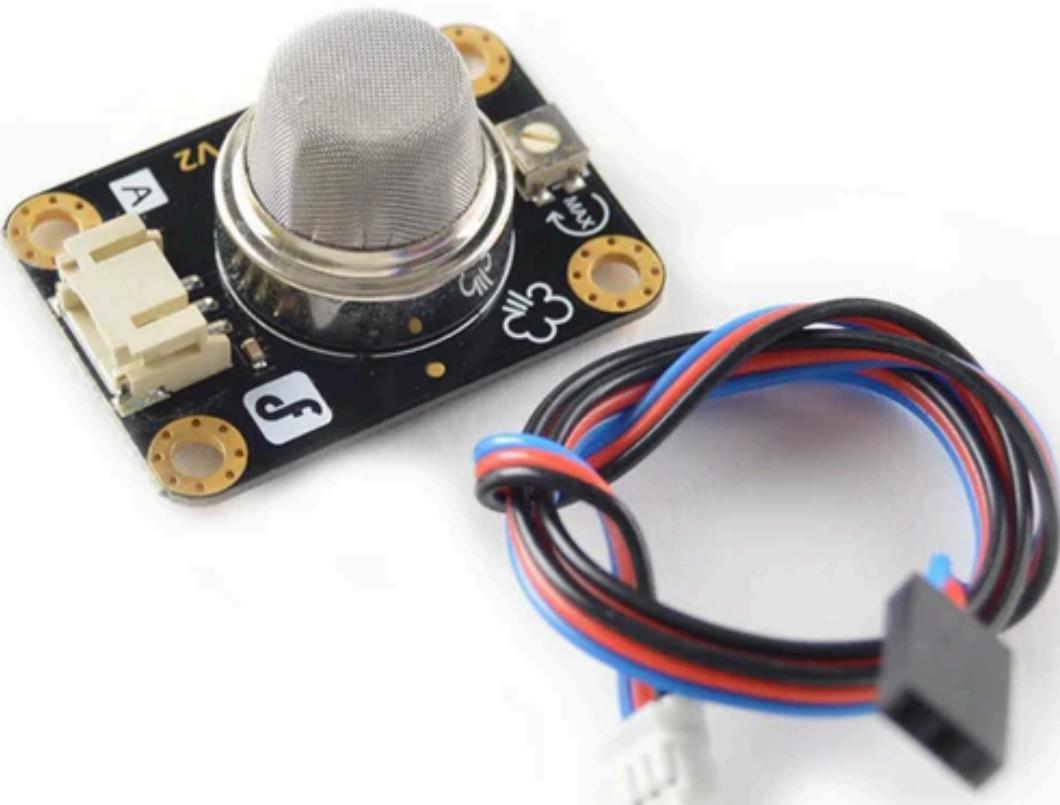
Le MCP3008 est utilisé dans diverses applications, telles que :

- Mesure des Signaux Analogiques : Il permet de lire des données de capteurs analogiques comme des thermistances, des photorésistances, des capteurs de gaz, etc.
- Interfaçage avec des Microcontrôleurs : Grâce au protocole SPI, il est facile à intégrer avec des microcontrôleurs et des micro-ordinateurs comme le Raspberry Pi ou Arduino.
- Applications d'Automatisation : Dans les projets de domotique ou de contrôle industriel, il permet de surveiller les signaux analogiques.

Le MCP3008 est un composant essentiel pour les projets qui nécessitent la conversion de signaux analogiques en valeurs numériques et leur intégration avec des systèmes numériques.

2.Capteur Gaz

Dans cette section, j'ai intégré un capteur MQ2 à notre casque connecté. Il est capable de détecter plusieurs types de gaz et, en raison de sa taille compacte, c'est un choix idéal pour ce type d'application.



Description Générale

Le capteur de gaz MQ-2 est un capteur polyvalent conçu pour détecter plusieurs types de gaz inflammables, y compris le méthane, le butane, le propane, le monoxyde de carbone, et la fumée. Il est largement utilisé dans des applications comme les systèmes de sécurité résidentiels, les détecteurs de gaz, et les systèmes de détection de fumée.

Principe de Fonctionnement

Le MQ-2 utilise un matériau sensible qui change de résistance lorsqu'il est exposé à certains gaz. Ce matériau, souvent basé sur des composés métalliques comme l'oxyde d'étain (SnO_2), possède une résistance qui varie en fonction de la concentration de gaz détectée. Le principe de base consiste à appliquer une tension au matériau sensible et à mesurer la résistance résultante. Plus la concentration de gaz est élevée, plus la résistance change de manière significative.

Caractéristiques Techniques

- **Alimentation** : Le MQ-2 fonctionne généralement avec une tension de 5V DC.
- **Sensibilité** : Il est sensible à plusieurs gaz inflammables, avec une plage de détection variable. La sensibilité peut être ajustée via un potentiomètre sur le module.
- **Plage de Détection** : Le MQ-2 peut détecter des concentrations de gaz allant de 300 ppm (parties par million) à 10 000 ppm.
- **Temps de Réponse** : Le capteur répond rapidement, avec un temps de réponse typique de moins de 10 secondes. Cela le rend approprié pour les applications nécessitant des alertes rapides en cas de détection de gaz ou de fumée.
- **Durée de Vie** : Le MQ-2 a une durée de vie relativement longue, généralement autour de 10 ans, mais cela dépend des conditions d'utilisation.
- **Température de Fonctionnement** : Il fonctionne correctement dans une plage de température de -20 °C à 70 °C.

Connexion et Utilisation

Le MQ-2 est souvent vendu sous forme de module prêt à l'emploi, ce qui facilite son intégration dans divers systèmes. Le module comprend généralement le capteur lui-même, un amplificateur, un potentiomètre pour régler la sensibilité, et des broches pour la connexion. Voici les connexions typiques :

- **VCC** : Alimentation de 5V DC.
- **GND** : Connexion à la masse (terre).
- **DO** : Sortie numérique, indiquant si la concentration de gaz a dépassé le seuil défini par le potentiomètre.
- **AO** : Sortie analogique, donnant une mesure continue de la concentration de gaz.

L'utilisation du MQ-2 nécessite généralement un calibrage pour garantir une sensibilité et une précision adéquates. Cela implique de régler le potentiomètre pour ajuster le seuil de déclenchement du signal numérique.

Applications Courantes

Le capteur de gaz MQ-2 est utilisé dans diverses applications, notamment :

- **Détecteurs de Gaz Résidentiels** : Pour alerter en cas de fuite de gaz domestique, comme le gaz naturel ou le propane.
- **Systèmes de Détection de Fumée** : Pour détecter la fumée provenant d'incendies ou de combustion.
- **Dispositifs de Sécurité Industriels** : Pour surveiller les niveaux de gaz dans les environnements industriels.
- **Projets Éducatifs** : Le MQ-2 est souvent utilisé dans des projets électroniques éducatifs ou des applications d'apprentissage, étant donné sa polyvalence et sa facilité d'utilisation.

Considérations de Sécurité

Lors de l'utilisation du capteur de gaz MQ-2, certaines précautions doivent être prises pour assurer la sécurité :

- **Ventilation** : Assurez-vous que l'environnement est bien ventilé lors de l'utilisation du capteur pour éviter l'accumulation de gaz.
- **Protection** : Le capteur doit être protégé des chocs et des dommages physiques.
- **Calibrage Régulier** : Pour des performances optimales, il est recommandé de calibrer régulièrement le capteur.
- **Température et Humidité** : Respectez les limites de température et d'humidité pour garantir un fonctionnement fiable.

Test capteur avec le convertisseur Analogique

Le capteur donne une tension de 0,2V lorsqu'il n'y a pas de gaz détecté. Si la tension dépasse 1V, cela indique la présence de gaz, suggérant que la résistance du capteur diminue avec l'augmentation de la concentration de gaz, permettant ainsi au courant de passer plus facilement.

Le convertisseur analogique/digital numérise cette tension en utilisant une résolution de 10 bits (jusqu'à 1023 valeurs différentes). Cela signifie que la valeur de sortie numérique peut atteindre 645 lorsqu'il y a une présence de gaz, ce qui correspond à une tension d'environ 1V. Pour une lecture indicative, une tension d'environ 0,5V pourrait suggérer la présence de gaz, indiquant une concentration de gaz significative.

Programmation Python

```
1 import RPi.GPIO as GPIO
2 import time
3
4 # Définir le pin auquel est connecté le capteur de gaz
5 gas_pin = 8
6
7 # Initialiser la bibliothèque GPIO
8 GPIO.setmode(GPIO.BOARD)
9 GPIO.setup(gas_pin, GPIO.IN)
10
11 try:
12     while True:
13         # Lire la valeur du capteur de gaz
14         gas_value = GPIO.input(gas_pin)
15
16         # Afficher le taux de gaz
17         print("Taux de gaz : {}".format(gas_value))
18
19         # Attendre 5 secondes avant la prochaine lecture
20         time.sleep(5)
21
22 except KeyboardInterrupt:
23     pass
24
25 # Réinitialiser les réglages GPIO
26 GPIO.cleanup()
27
```

Bibliothèques Importées

- RPi.GPIO: C'est la bibliothèque utilisée pour contrôler les broches GPIO du Raspberry Pi. Elle vous permet de configurer les broches comme entrées ou sorties, et de lire ou écrire des valeurs.
- time: Fournit des fonctions pour travailler avec le temps, comme sleep, qui permet de faire des pauses dans le code.

Configuration du Capteur de Gaz

- gas_pin = 8 : Définit la broche GPIO à laquelle le capteur de gaz est connecté. Dans ce cas, le capteur est branché à la broche 8.

- GPIO.setmode(GPIO.BOARD) : Configure le mode de numérotation des broches du Raspberry Pi. "BOARD" signifie que les broches sont numérotées selon leur position physique sur le Raspberry Pi (et non pas selon la numérotation GPIO spécifique au modèle).
- BOARD:** Les broches sont numérotées de 1 à 40 (ou 26, selon le modèle de Raspberry Pi), en fonction de leur emplacement physique sur le connecteur. Par exemple, la broche 1 est toujours la première broche du connecteur, et la broche 2 la deuxième, indépendamment de leur fonction interne ou de leur lien avec le chipset Broadcom.

-
- GPIO.setup(gas_pin, GPIO.IN) : Configure la broche 8 comme une entrée, car le capteur de gaz envoie des signaux au Raspberry Pi.

Boucle de Lecture et Affichage

- try: ouvre un bloc d'essai pour gérer les interruptions du programme (comme un appui sur Ctrl+C).
- while True: crée une boucle infinie qui continue jusqu'à ce qu'elle soit interrompue.
 - gas_value = GPIO.input(gas_pin): Lit la valeur du capteur de gaz à partir de la broche définie. Le résultat sera généralement 0 ou 1, indiquant si le seuil du capteur est dépassé ou non.
 - print("Taux de gaz : {}".format(gas_value)) : Affiche la valeur du capteur de gaz sur la console.
 - time.sleep(5): Attend 5 secondes avant de refaire une lecture, ce qui évite de surcharger le système avec des lectures continues.

Gestion des Interruptions et Nettoyage

- except KeyboardInterrupt: capture les interruptions clavier, comme lorsque vous appuyez sur Ctrl+C. Cela permet d'arrêter proprement la boucle infinie.
- GPIO.cleanup(): Nettoie les configurations GPIO, réinitialisant les broches à leur état par défaut. Cela évite des problèmes de configuration si d'autres programmes utilisent les broches GPIO après ce code.

Explication simple :

Ce code configure une broche GPIO comme entrée pour lire les données d'un capteur de gaz. Il utilise une boucle infinie pour lire et afficher les valeurs du capteur toutes les 5 secondes. La gestion des interruptions permet un arrêt propre du programme, et le nettoyage des GPIO garantit que les ressources du Raspberry Pi sont correctement réinitialisées.

3. Capteur Température et humidité

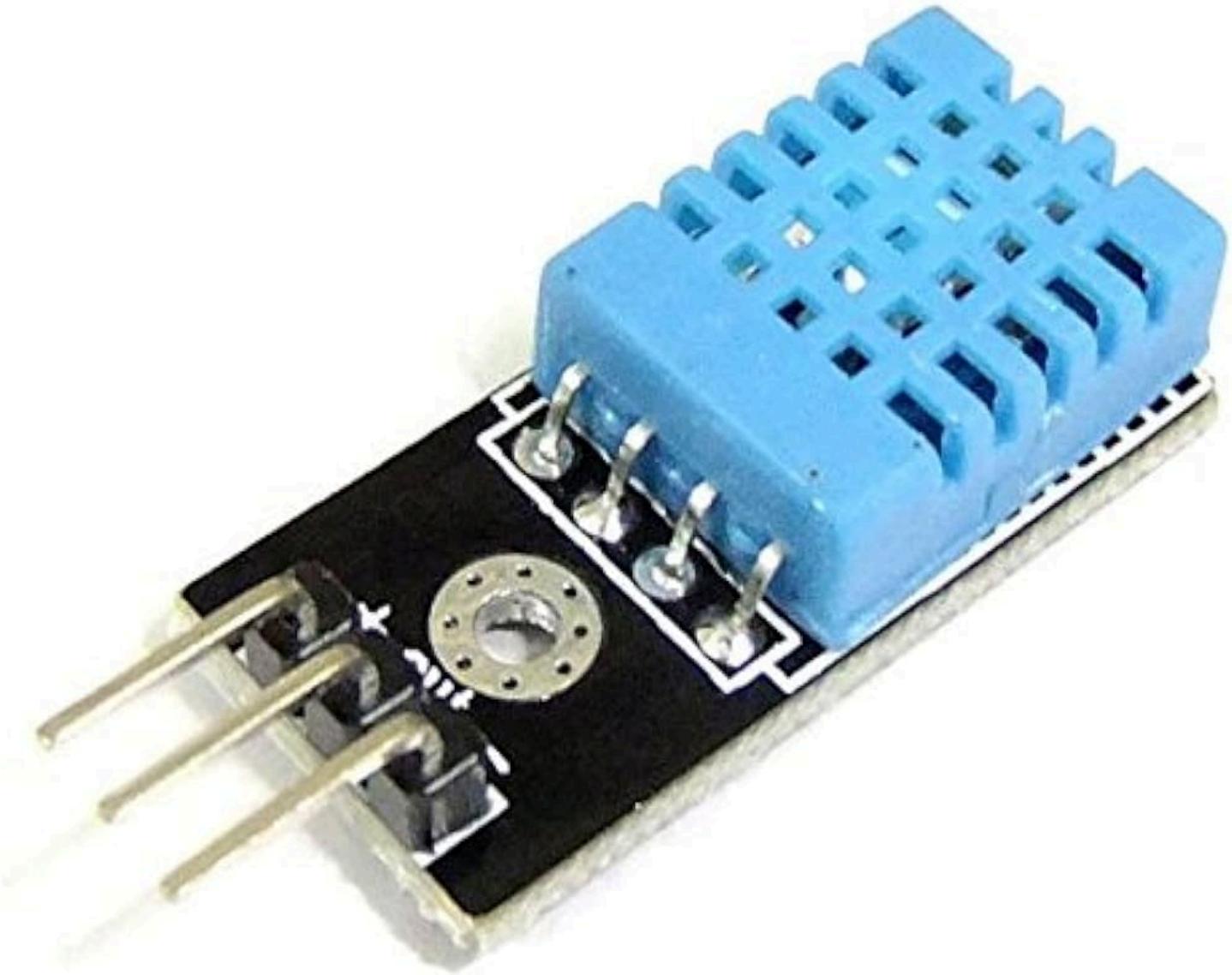
Description du Capteur DHT11

Le capteur DHT11 est un dispositif largement utilisé pour mesurer à la fois la température et l'humidité relative de l'air ambiant. Il est particulièrement apprécié pour sa facilité d'utilisation, son faible coût, et sa fiabilité dans des applications nécessitant des mesures non critiques.

Caractéristiques principales:

- **Plage de mesure de température:** 0 à 50 °C avec une précision de ± 2 °C.
- **Plage de mesure d'humidité:** 20% à 80% RH (Relative Humidity) avec une précision de $\pm 5\%$ RH.
- **Alimentation:** 3 à 5,5 Volts.
- **Interface de communication:** Numérique, via un seul fil de données (Single-bus).

Le DHT11 intègre un capteur d'humidité capacitif et un thermistor pour mesurer l'air ambiant, et un petit microcontrôleur qui convertit les lectures analogiques en valeurs numériques. Les données sont ensuite transmises numériquement via un protocole spécifique à DHT.

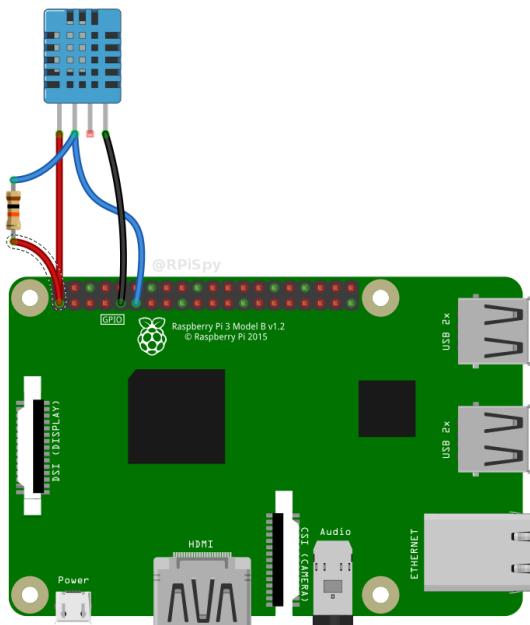


Fonctionnement du Capteur DHT11

Le fonctionnement du DHT11 peut être décrit par les étapes suivantes:

- Initialisation:** Le microcontrôleur envoie un signal de démarrage pour activer le capteur.
- Mesure et conversion:** Le capteur mesure l'humidité et la température et les convertit en signaux électriques analogiques. Ces signaux sont convertis en données numériques par le microcontrôleur interne.
- Transmission des données:** Les données numériques sont transmises via une communication en série sur un seul fil.

Schéma de Connexion



Explication du Schéma

- **+V et GND** : Connexion de l'alimentation au capteur.
- **Data Pin** : Connecté à un GPIO du microcontrôleur, avec une résistance de pull-up entre +V et le pin de données pour stabiliser le signal.
- **Résistance de pull-up** : Essentielle pour la communication fiable sur le bus de données unique.

Utilisation

Pour utiliser le DHT11, il faut initier une lecture à partir du microcontrôleur hôte. Le capteur transmettra une séquence de 40 bits qui contient les informations de l'humidité et de la température. Il est important de respecter les timings spécifiés dans la datasheet du capteur pour une communication réussie.

En conclusion, le capteur DHT11 est idéal pour des projets nécessitant des mesures de base de la température et de l'humidité de l'air. Il est couramment utilisé dans les applications de maison intelligente, les systèmes de contrôle de l'environnement, et les projets éducatifs liés à l'Arduino ou à d'autres microcontrôleurs.

Programmation Python

```
1 import Adafruit_DHT
2 sensor = Adafruit_DHT.DHT11
3 pin = 4
4 humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
5
6 print('Test')
7
8 if humidity is not None and temperature is not None:
9     print('Temp={0:0.1f}*C  Humidity={1:0.1f}%' .format(temperature, humidity))
10 else:
11     print('Failed to get reading. Try again!')
12
```

Bibliothèques Importées

Adafruit_DHT: Cette bibliothèque est utilisée pour interagir avec les capteurs DHT, tels que le DHT11 et le DHT22. Elle permet de lire les données de température et d'humidité fournies par ces capteurs.

Configuration du Capteur DHT11

sensor = Adafruit_DHT.DHT11: Définit le type de capteur utilisé. Dans ce cas, c'est un DHT11. Cela permet à la bibliothèque de savoir comment communiquer avec le capteur pour lire les données.

pin = 4: Définit la broche GPIO à laquelle le capteur est connecté. Ici, le capteur DHT11 est connecté à la broche GPIO 4 du microcontrôleur (par exemple, un Raspberry Pi).

Lecture des Données et Affichage

- **Adafruit_DHT.read_retry(sensor, pin):** Cette fonction tente de lire les données du capteur plusieurs fois pour s'assurer d'obtenir une lecture correcte. Elle retourne deux valeurs : l'humidité et la température.
- **print('Test'):** Imprime le mot "Test" dans la console pour indiquer que le script s'exécute correctement jusqu'à ce point.

- **if humidity is not None and temperature is not None:** Vérifie si les valeurs de l'humidité et de la température sont valides (c'est-à-dire qu'elles ne sont pas None).
- `*print('Temp={0:0.1f}C Humidity={1:0.1f}%'.format(temperature, humidity))`: Si les valeurs sont valides, elles sont formatées et imprimées. La température est affichée avec une précision d'une décimale et suivie de *C pour indiquer les degrés Celsius. L'humidité est également affichée avec une précision d'une décimale et suivie de % pour indiquer le pourcentage d'humidité.
- **else:** Si les valeurs ne sont pas valides (c'est-à-dire si la lecture a échoué), un message d'erreur "Failed to get reading. Try again!" est affiché.

Explication Simple

Ce code configure un capteur DHT11 pour lire la température et l'humidité. Il vérifie si les lectures sont valides, puis les affiche dans la console. Si la lecture échoue, un message d'erreur est imprimé. Cela permet de surveiller les conditions environnementales en temps réel en utilisant un microcontrôleur comme un Raspberry Pi.

Programmation Python

Compilation des deux capteurs

Annexe 1 E2

1. Boucle Principale :

```
try:  
    while True:  
        # Lecture du capteur de gaz  
        gas_value = readadc(gas_sensor, CLK, DOUT, DIN, CS)  
        print("Taux de gaz : {}".format(gas_value))
```

- **readadc(gas_sensor, CLK, DOUT, DIN, CS)**: Cette fonction lit la valeur du capteur de gaz connecté via les broches spécifiées (CLK, DOUT, DIN, CS). gas_sensor désigne le capteur de gaz.
- **print("Taux de gaz : {}".format(gas_value))**: Affiche la valeur lue du capteur de gaz.

```
# Lecture du capteur de température et d'humidité  
humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)  
  
if humidity is not None and temperature is not None:  
    print('Temp={0:0.1f}*C  Humidity={1:0.1f}%'.format(temperature, humidity))
```

- **Adafruit_DHT.read_retry(sensor, pin)**: Lit les données du capteur de température et d'humidité. sensor spécifie le type de capteur (par exemple, DHT11), et pin spécifie le GPIO auquel le capteur est connecté.

- **if humidity is not None and temperature is not None:** Vérifie que les valeurs lues sont valides (pas None).
- ***print('Temp={0:0.1f}C Humidity={1:0.1f}%'.format(temperature, humidity)):** Affiche les valeurs de température et d'humidité.

2. Insertion des Données dans les Bases de Données :

```
# Insertion des données dans la base de données locale
sql_insert_local = "INSERT INTO ouvrier1 (temperature, humidite, tauxgaz) VALUES (%s, %s, %s)"
val_local = (temperature, humidity, gas_value)
cursor_local.execute(sql_insert_local, val_local)
conn_local.commit()
print("Données insérées dans la base de données locale.")
```

- **sql_insert_local:** La requête SQL pour insérer les données dans une table appelée ouvrier1 dans la base de données locale.
- **val_local:** Les valeurs de température, d'humidité et de taux de gaz à insérer.
- **cursor_local.execute(sql_insert_local, val_local):** Exécute la requête d'insertion avec les valeurs fournies.
- **conn_local.commit():** Confirme l'insertion des données dans la base de données locale.
- **print("Données insérées dans la base de données locale."):** Affiche un message confirmant l'insertion des données.

```
# Insertion des données dans la base de données distante
sql_insert_distante = "INSERT INTO ouvrier1 (temperature, humidite, tauxgaz) VALUES (%s, %s, %s)"
val_distante = (temperature, humidity, gas_value)
cursor_distante.execute(sql_insert_distante, val_distante)
conn_distante.commit()
print("Données insérées dans la base de données distante.")
```

- **sql_insert_distante:** La requête SQL pour insérer les données dans une table ouvrier1 dans la base de données distante.
- **val_distante:** Les valeurs de température, d'humidité et de taux de gaz à insérer.
- **cursor_distante.execute(sql_insert_distante, val_distante):** Exécute la requête d'insertion avec les valeurs fournies.

- **conn_distante.commit()**: Confirme l'insertion des données dans la base de données distante.
- **print("Données insérées dans la base de données distante.")**: Affiche un message confirmant l'insertion des données.

3. Gestion des Erreurs et Attente :

```

    else:
        print('Échec de la lecture. Réessayer!')

        time.sleep(5)

```

- **else**: Exécuté si les valeurs lues sont None (échec de lecture).
- **print('Échec de la lecture. Réessayer!')**: Affiche un message d'erreur.
- **time.sleep(5)**: Attend 5 secondes avant de refaire une lecture pour éviter une surcharge du système.

4. Gestion des Interruptions et Nettoyage :

```

except KeyboardInterrupt:
    GPIO.cleanup()
    conn_local.close()
    conn_distante.close()

```

- **except KeyboardInterrupt**: Capture les interruptions clavier (comme Ctrl+C) pour permettre un arrêt propre du programme.
- **GPIO.cleanup()**: Réinitialise les broches GPIO à leur état par défaut.
- **conn_local.close()**: Ferme la connexion à la base de données locale.
- **conn_distante.close()**: Ferme la connexion à la base de données distante.

Conclusion

Ce code lit en continu les données d'un capteur de gaz et d'un capteur de température et d'humidité, affiche ces valeurs et les insère dans deux bases de données (une locale et une distante). Il gère les erreurs de lecture et attend 5 secondes entre chaque lecture pour éviter une surcharge. En cas d'interruption du programme, il ferme proprement les connexions et réinitialise les broches GPIO.

Envoie des données (Annexe 1 E2)

1. Bibliothèques Importées

```
import RPi.GPIO as GPIO
import time
import mysql.connector
```

- **RPi.GPIO:** Cette bibliothèque permet de contrôler les broches GPIO du Raspberry Pi.
- **time:** Fournit des fonctions pour travailler avec le temps, telles que sleep.
- **mysql.connector:** Permet de se connecter et d'interagir avec une base de données MySQL.

2. Initialisation des Broches GPIO

```
# Initialisation des broches GPIO
GPIO.setmode(GPIO.BCM)
CLK  = 11
DOUT = 9
DIN  = 10
CS   = 8
GPIO.setup(CLK,  GPIO.OUT)
GPIO.setup(DOUT, GPIO.IN)
GPIO.setup(DIN,  GPIO.OUT)
GPIO.setup(CS,   GPIO.OUT)
```

- **GPIO.setmode(GPIO.BCM):** Définit le mode de numérotation des broches en utilisant le schéma de numérotation BCM (Broadcom SOC Channel).
- **CLK, DOUT, DIN, CS:** Affecte les broches GPIO à des variables spécifiques pour le convertisseur analogique-numérique (ADC).

- **GPIO.setup:** Configure les broches comme sorties (OUT) ou entrées (IN) selon leurs rôles respectifs dans la communication avec le capteur.

3. Connexion à la Base de Données MySQL

```
# Connexion à la base de données MySQL
✓ conn = mysql.connector.connect(
    host="172.16.112.193",
    user="root",
    password="",
    database="gaz"
)
```

- **mysql.connector.connect:** Établit une connexion à la base de données MySQL. Les paramètres incluent l'adresse IP du serveur (host), le nom d'utilisateur (user), le mot de passe (password), et le nom de la base de données (database).
- **cursor = conn.cursor():** Crée un curseur pour exécuter des commandes SQL.

4 Fonction readadc

```
✓ def readadc(num, clk, dout, din, cs):
    # Lecture ADC (Analog to Digital Converter)
    if ((num > 7) or (num < 0)):
        return -1
```

- **readadc:** Cette fonction lit les données d'un canal spécifique de l'ADC.
- **if ((num > 7) or (num < 0)): return -1:** Vérifie si le numéro du canal est valide (entre 0 et 7). Si non, retourne -1.

```

    GPIO.output(cs, 1) # Arrêt Clock
    GPIO.output(clk, 0) # start clock
    GPIO.output(cs, 0) # Sélection du canal à débuter

```

- **GPIO.output(cs, 1)**: Met la broche CS (Chip Select) à l'état haut pour désactiver le canal.
- **GPIO.output(clk, 0)**: Initialise la broche de l'horloge à l'état bas.
- **GPIO.output(cs, 0)**: Met la broche CS à l'état bas pour sélectionner le canal et démarrer la communication.

```

command = num
command |= 0x18
command <<= 3

```

- **command**: Prépare la commande pour l'ADC. Le numéro du canal est combiné avec les bits de configuration (0x18) et décalé vers la gauche pour former une commande de 8 bits.

```

for i in range(5):
    if (command & 0x80):
        GPIO.output(din, 1)
    else:
        GPIO.output(din, 0)

    command <<= 1
    GPIO.output(clk, 1) # Horloge impulsion mise en action
    GPIO.output(clk, 0)

```

- **for i in range(5)**: Envoie les 5 bits de la commande à l'ADC, un bit à la fois.
- **GPIO.output(din, 1/0)**: Définit la broche DIN à l'état haut ou bas selon le bit actuel de la commande.
- **GPIO.output(clk, 1/0)**: Crée une impulsion d'horloge pour envoyer le bit à l'ADC.

```

out = 0

for i in range(12):
    GPIO.output(clk, 1) # Horloge impulsion mise en action
    GPIO.output(clk, 0)
    out <<= 1
    out |= GPIO.input(dout)

```

- **out = 0:** Initialise la variable de sortie qui contiendra la valeur lue.
- **for i in range(12):** Lit les 12 bits de données de l'ADC, un bit à la fois.
- **GPIO.output(clk, 1/0):** Crée une impulsion d'horloge pour lire chaque bit.
- **out <<= 1:** Décale la

valeur lue vers la gauche pour faire de la place pour le nouveau bit.

- **out |= GPIO.input(dout):** Lit le bit actuel de la broche DOUT et l'ajoute à la valeur de sortie.

```

GPIO.output(cs, 1) # Arrêt de la connection avec le capteur de gaz via le CAN.
out >>= 1
return out

```

- **GPIO.output(cs, 1):** Met la broche CS à l'état haut pour terminer la communication avec le capteur.
- **out >>= 1:** Décale la valeur lue vers la droite pour ajuster les bits (car l'ADC donne un bit supplémentaire de sortie).
- **return out:** Retourne la valeur lue de l'ADC, qui correspond à la valeur analogique du capteur de gaz convertie en numérique.

Conclusion

Ce code permet de lire les données d'un capteur de gaz via un convertisseur analogique-numérique (ADC) connecté à un Raspberry Pi. Les données sont ensuite insérées dans une base de données MySQL. Voici un résumé des étapes clés :

1. **Initialisation des broches GPIO** : Configure les broches nécessaires pour la communication avec l'ADC.
2. **Connexion à la base de données MySQL** : Établit une connexion à la base de données pour stocker les données lues.
3. **Définition du canal du capteur de gaz** : Spécifie quel canal de l'ADC est utilisé pour lire les données du capteur de gaz.
4. **Fonction readadc** :
 - Vérifie la validité du canal.
 - Prépare et envoie une commande à l'ADC pour sélectionner le canal et démarrer la conversion.
 - Lit les 12 bits de données de l'ADC.
 - Retourne la valeur lue.

Ce code doit être intégré dans une boucle continue pour lire les données périodiquement et les insérer dans la base de données, ou être appelé dans une fonction plus grande qui gère le timing et l'insertion des données. Assurez-vous également de gérer les erreurs de connexion à la base de données et de nettoyer correctement les broches GPIO à la fin de l'exécution du programme.

Travail Etudiant n°3

1. Capteur Grove GPS

J'ai utilisé un capteur Grove GPS, et voici quelques informations sur le capteur :

Le capteur Grove GPS, basé sur le module GPS standard NEO-6, utilise l'interface UART pour communiquer avec le Raspberry Pi.

Le capteur Grove GPS envoie des informations de positionnement global en utilisant des protocoles standard comme NMEA (National Marine Electronics Association).

Fonctionnement du capteur GPS

1. Réception des signaux satellites:

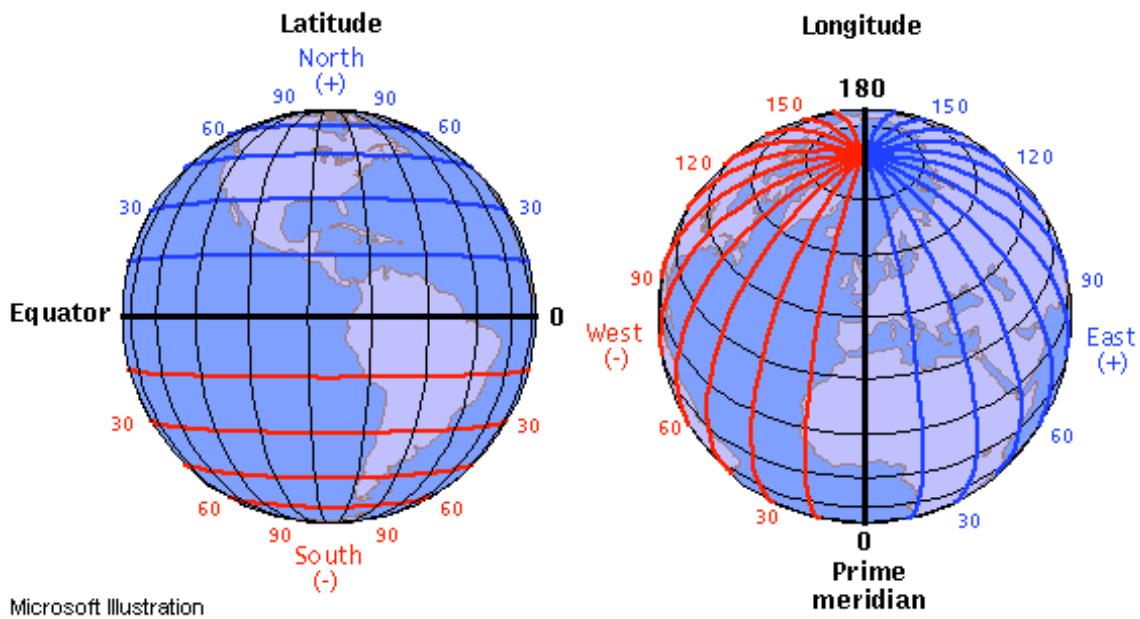
- Le capteur GPS capte les signaux provenant d'au moins quatre satellites GPS. Chaque signal contient des informations sur l'heure d'émission et la position du satellite au moment de l'émission.

2. Calcul de la position:

- En utilisant la différence de temps entre l'émission du signal par le satellite et la réception de ce signal, le module GPS peut calculer la distance à chaque satellite.
- Avec les distances de quatre satellites ou plus, le capteur utilise une méthode appelée trilateration pour calculer sa position précise (latitude, longitude, altitude).

3. Format NMEA:

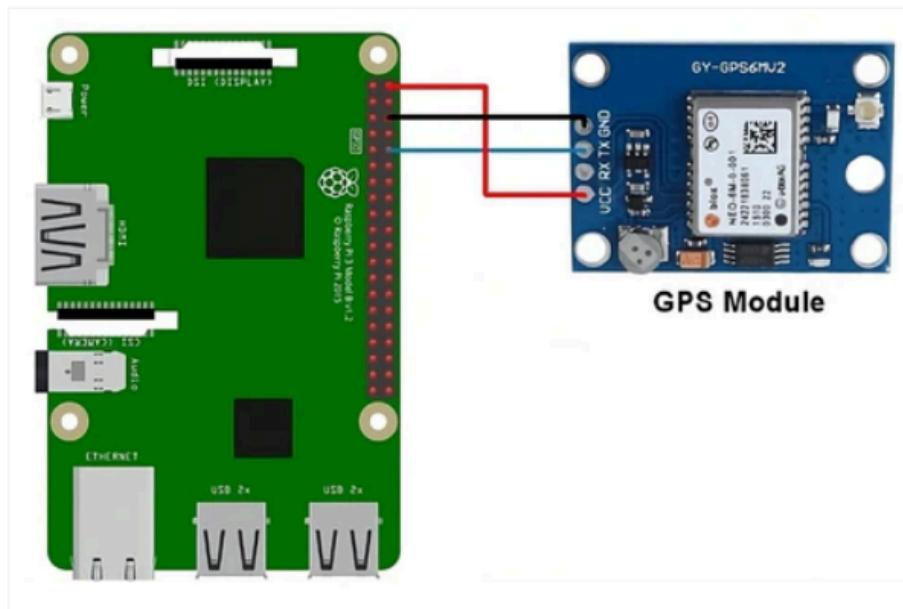
- Les informations calculées sont formatées en messages NMEA, qui sont des phrases ASCII standardisées contenant des informations sur la position, l'heure, la vitesse, etc.
- Les phrases NMEA courantes comprennent \$GPGGA, \$GPRMC, etc.



Microsoft Illustration

Le capteur GPS transmet les données via une connexion série UART à une vitesse de baud standard (9600 bauds).

J'ai utilisé un capteur de Grove GPS et je l'ai directement connecté au Raspberry Pi.



Pour connecter le module GPS à un Raspberry Pi 3 il faut connecter :

- **Alimentation** : Le VCC du module GPS est connecté au 5V du Raspberry Pi pour fournir l'énergie nécessaire.
- **Masse** : Le GND du GPS est relié au GND du Raspberry Pi pour établir une référence commune.
- **Transmission des données** : Le TX du GPS est connecté à la broche GPIO15 (RXD) du Raspberry Pi, permettant la réception des données de localisation.

Ces connexions permettent au Raspberry Pi de recevoir des données GPS précises.

Après cela il faut installer toutes les bibliothèques du grove gps avec la commande lib "install Seeed_Arduino_GroveGPS" Cette commande installera toutes les bibliothèques requises pour faire fonctionner le module Grove GPS avec votre Raspberry Pi 3.

2. Récupération, Traitement et stockage des données du GPS

Programme en Python pour la récupération, traitement et stockage des données du GPS

```
3 # Import des modules nécessaires
4 import serial # Module pour la communication série
5 import pynmea2 # Module pour le traitement des données NMEA
6 import mysql.connector # Module pour la connexion à la base de données MySQL/MariaDB
7 import time # Module pour la gestion du temps
8
```

1. Importation des modules

- serial : Utilisé pour la communication série avec le module GPS.
- pynmea2 : Utilisé pour le traitement des données NMEA, le format standard des données GPS.
- mysql.connector : Utilisé pour se connecter à une base de données MySQL/MariaDB.
- time : Utilisé pour la gestion du temps et des délais d'attente.

```

6      tabnine: test | explain | document | ask
7
8  def connect_local_database():
9      # Fonction pour se connecter à la base de données locale MariaDB
10     return mysql.connector.connect(
11         host="localhost", # Adresse IP ou nom d'hôte du serveur MariaDB Local
12         user="root", # Nom d'utilisateur de la base de données MariaDB
13         password="pi", # Mot de passe de la base de données MariaDB
14         database="casque", # Nom de la base de données MariaDB
15         auth_plugin='mysql_native_password'
16     )
17
18
19  tabnine: test | explain | document | ask
20  def connect_remote_database():
21      # Fonction pour se connecter à la base de données distante MariaDB
22      return mysql.connector.connect(
23          host="172.16.112.193", # Adresse IP ou nom d'hôte du serveur MariaDB distant
24          user="root", # Nom d'utilisateur de la base de données MariaDB
25          password="", # Mot de passe de la base de données MariaDB
26          database="casque", # Nom de la base de données MariaDB
27          auth_plugin='mysql_native_password'
28      )

```

2. Connexion à la base de données

a. Connexion à la base de données locale

Cette fonction établit une connexion avec une base de données locale MariaDB, utilisant les informations de connexion spécifiées.

b. Connexion à la base de données distante

Cette fonction établit une connexion avec une base de données distante MariaDB, en utilisant les informations de connexion fournies.

```

28
29  tabnine: test | explain | document | ask
30  def read_gps_data(db_connection, db_cursor):
31      # Fonction pour lire les données GPS
32      serial_port = serial.Serial("/dev/ttyS0", 9600, timeout=0.5) # Configuration du port série
33
34      try:
35          while True:
36              start_time = time.time() # Temps de début de la boucle
37
38              raw_data = serial_port.readline().decode("utf-8").strip() # Lecture des données brutes du GPS
39              print("Raw Data:", raw_data) # Affichage de la trame NMEA sans les caractères de nouvelle ligne
40
41              try:
42                  nmea_data = pynmea2.parse(raw_data) # Analyse de la trame NMEA
43
44                  if isinstance(nmea_data, pynmea2.GGA): # Vérification du type de trame (GPGGA)
45                      if nmea_data.gps_qual == 1: # Vérification de la qualité du fix GPS
46                          latitude = nmea_data.latitude # Récupération de la Latitude
47                          longitude = nmea_data.longitude # Récupération de la Longitude
48
49                          # Insertion des données dans la base de données
50                          sql_insert_query = "INSERT INTO ouvrier1 (latitude, longitude) VALUES (%s, %s)"
51                          insert_values = (latitude, longitude)
52                          print("Insert Values:", insert_values) # Affichage des valeurs à insérer dans la base de données
53                          db_cursor.execute(sql_insert_query, insert_values)
54                          db_connection.commit()
55
56                          print("Données insérées dans la base de données.")

```

3. Lecture des données GPS

Cette fonction configure le port série pour lire les données GPS provenant du module connecté.

a. Analyse des données NMEA

Les données brutes sont analysées à l'aide de pynmea2 pour extraire des informations GPS utilisables.

b. Vérification et extraction des données GPS

```
if isinstance(nmea_data, pynmea2.GGA):

    if nmea_data.gps_qual == 1:

        latitude = nmea_data.latitude

        longitude = nmea_data.longitude

        sql_insert_query = "INSERT INTO ouvrier1 (latitude, longitude) VALUES (%s, %s)"

        insert_values = (latitude, longitude)

        print("Insert Values:", insert_values)

        db_cursor.execute(sql_insert_query, insert_values)

        db_connection.commit()

        print("Données insérées dans la base de données.")
```

Le code vérifie si les données GPS sont valides (type GGA et qualité de fix GPS) avant de les insérer dans la base de données.

```

70
71 if __name__ == "__main__":
72     # Connexion à la base de données Locale
73     local_db_connection = connect_local_database()
74     local_db_cursor = local_db_connection.cursor()
75
76     # Lecture des données GPS et stockage dans La base de données Locale
77     read_gps_data(local_db_connection, local_db_cursor)
78
79     # Connexion à la base de données distante
80     remote_db_connection = connect_remote_database()
81     remote_db_cursor = remote_db_connection.cursor()
82
83     # Lecture des données GPS et stockage dans La base de données distante
84     read_gps_data(remote_db_connection, remote_db_cursor)
85

```

4. Exécution principale

```

if __name__ == "__main__":
    local_db_connection = connect_local_database()
    local_db_cursor = local_db_connection.cursor()

    read_gps_data(local_db_connection, local_db_cursor)

    remote_db_connection = connect_remote_database()
    remote_db_cursor = remote_db_connection.cursor()

    read_gps_data(remote_db_connection, remote_db_cursor)

```

Le bloc principal du code établit des connexions aux bases de données locale et distante, puis lance la lecture des données GPS pour chaque connexion.

3. Stockage des données des capteur(gaz, température et humidité)

Pour cette étape, j'ai demandé à mon collègue de combiner le code pour gérer les capteurs de gaz, de température et d'humidité dans un seul fichier. Ensuite, j'ai ajouté la bibliothèque mysql.connector pour établir une connexion à une base de données et y stocker les données collectées.

Connexion à la base de données locale MariaDB

```
conn_local = mysql.connector.connect(  
    host="localhost",  
    user="root",  
    password="pi",  
    database="casque",  
    auth_plugin='mysql_native_password'  
)
```

```
cursor_local = conn_local.cursor()
```

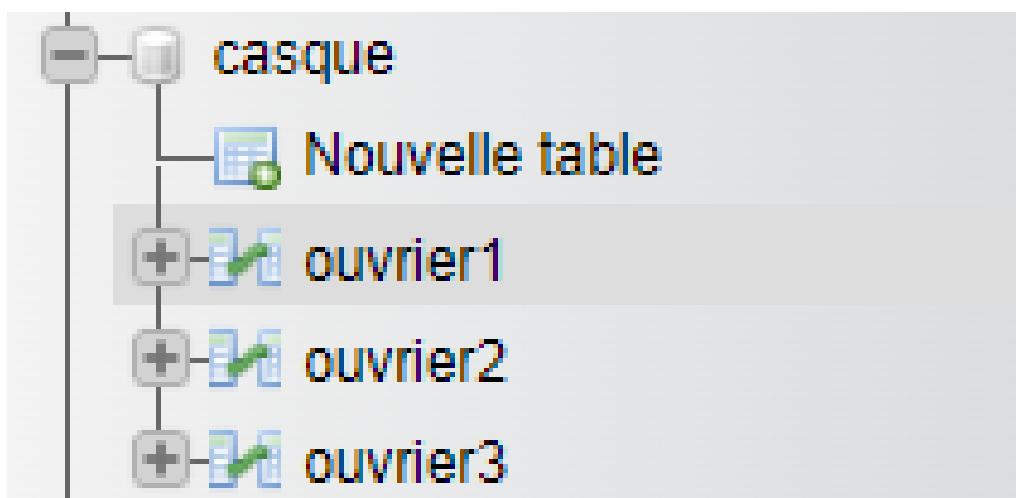
Connexion à la base de données distante MySQL (phpMyAdmin)

```
conn_distante = mysql.connector.connect(  
    host="172.16.112.193",  
    user="root",  
    password="\"",  
    database="casque",  
    auth_plugin='mysql_native_password'  
)
```

J'utilise deux bases de données, une locale et une distante, pour assurer une sécurité supplémentaire. Si la connexion à la base de données distante est interrompue, nous avons toujours la base de données locale pour sauvegarder les données.

4. Crédation du site de monitoring et la base de données

1. Base de données

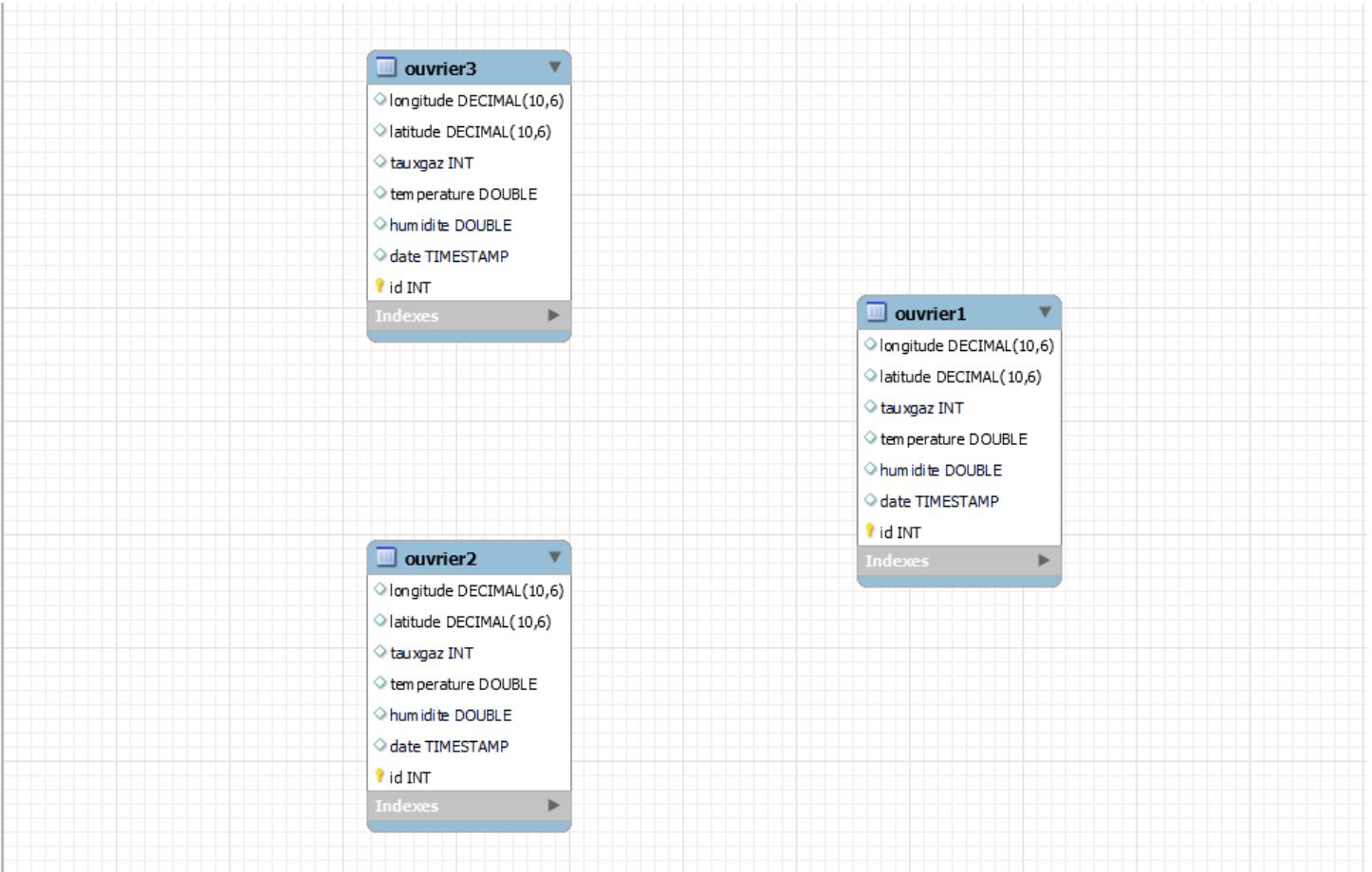


The screenshot shows the MySQL Workbench interface with the following details:

- Serveur : MySQL 3306
- Base de données : casque
- Table : ouvrier1
- Structure tab (selected)
- SQL tab
- Rechercher tab
- Insérer tab
- Exporter tab
- Importer tab
- Privilèges tab
- Opérations tab
- Déclencheurs tab

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	longitude	decimal(10,6)			Oui	NULL			Modifier Supprimer Plus
2	latitude	decimal(10,6)			Oui	NULL			Modifier Supprimer Plus
3	tauxgaz	int			Oui	NULL			Modifier Supprimer Plus
4	temperature	double			Oui	NULL			Modifier Supprimer Plus
5	humidite	double			Oui	NULL			Modifier Supprimer Plus
6	date	timestamp			Oui	CURRENT_TIMESTAMP	DEFAULT_GENERATED		Modifier Supprimer Plus
7	id	int			Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer Plus

Tout cocher Avec la sélection : [Parcourir](#) [Modifier](#) [Supprimer](#) [Primaire](#) [Unique](#) [Index](#) [Spatial](#) [Texte entier](#)



Pour la base de données, j'ai créé une base avec trois tables, une pour chaque casque qui sera utilisé. Si dans le futur on veut en rajouter, on peut le faire sans problème. Chaque table se compose d'une longitude, latitude, taux de gaz, température, humidité, date et ID.

- **longitude** : la longitude récupérée par le capteur GPS.
- **latitude** : la latitude récupérée par le capteur GPS.
- **Température** : La température mesurée par le capteur.
- **Humidité** : Le taux d'humidité enregistré par le capteur.
- **date** : la date à laquelle une nouvelle donnée est insérée dans la base de données.
- **ID** : Un identifiant auto-incrémenté pour chaque entrée dans la table.

2. Site de monitoring

Monitoring Chantier

Casque n°1 dernière utilisation : 2024-05-20 16:02:20

Casque n°2 dernière utilisation : jamais utilisé.

Casque n°3 dernière utilisation : jamais utilisé.

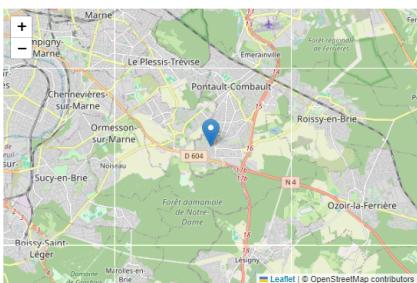
Historique des activités



Pour le site, j'ai fait une page d'accueil avec tous les casques avec leur dernière utilisation et un historique des événements pour voir tous les événements qui se sont produits dans la journée (température supérieure à 50°C ou un taux de gaz au-dessus de 300).

Monitoring Chantier

Ouvrier N°1



capteur de gaz : Pas de gaz présent.

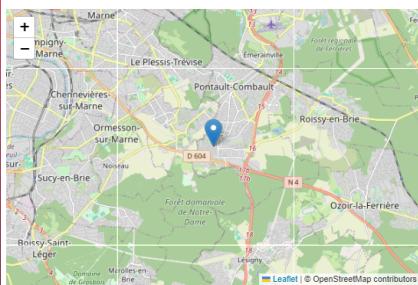
capteur de température :
Température normale : 23°C

Sur la page du casque n°1, on peut voir une carte indiquant l'emplacement de l'ouvrier en cas d'accident, la présence de gaz autour du casque, ainsi que la température ambiante actuelle.

Monitoring Chantier

Attention, Gaz Présent chez l'ouvrier n°1

Ouvrier N°1



capteur de gaz :

Attention GAZ présent!!!

capteur de température :

Attention forte température!! : 75°C

Si l'un des capteurs détecte une valeur qui dépasse leur seuil limite, des alertes seront générées, et l'alerte « Attention, Gaz Présent chez l'ouvrier n°1 » s'affichera en haut de la page, peu importe où l'on se trouve, car elle est attachée au header.

Monitoring Chantier

AAAA-MM-JJ |

retour

gaz ou haute température présent: Taux de Gaz : 342 | temperature: 21 - Date: 2024-05-23 09:41:31
gaz ou haute température présent: Taux de Gaz : 374 | temperature: 21 - Date: 2024-05-23 08:41:01
gaz ou haute température présent: Taux de Gaz : 373 | temperature: 21 - Date: 2024-05-23 08:40:56
gaz ou haute température présent: Taux de Gaz : 373 | temperature: 21 - Date: 2024-05-23 08:40:50
gaz ou haute température présent: Taux de Gaz : 373 | temperature: 21 - Date: 2024-05-23 08:40:45
gaz ou haute température présent: Taux de Gaz : 372 | temperature: 21 - Date: 2024-05-23 08:40:39
gaz ou haute température présent: Taux de Gaz : 372 | temperature: 21 - Date: 2024-05-23 08:40:34
gaz ou haute température présent: Taux de Gaz : 371 | temperature: 21 - Date: 2024-05-23 08:40:28
gaz ou haute température présent: Taux de Gaz : 407 | temperature: 21 - Date: 2024-05-23 08:40:23
gaz ou haute température présent: Taux de Gaz : 391 | temperature: 20 - Date: 2024-05-23 08:38:52
gaz ou haute température présent: Taux de Gaz : 300 | temperature: 75 - Date: 2024-05-20 16:02:20

Dans l'historique des événements, on peut voir toutes les alertes qui ont dépassé leur seuil. Elles sont affichées ici, triées par ordre croissant, et il y a une barre de recherche en haut pour simplifier la recherche d'une alerte grâce à sa date.

3. les codes pour le site.

code php de la page du casque n°1 :

```
1 <?php
2 // Connexion à la base de données
3 $servername = "localhost"; // Adresse du serveur MySQL
4 $username = "root"; // Nom d'utilisateur MySQL
5 $password = ""; // Mot de passe MySQL
6 $dbname = "casque"; // Nom de la base de données MySQL
7
8 // Créer une connexion
9 $conn = new mysqli($servername, $username, $password, $dbname);
10
11 // Vérifier la connexion
12 if ($conn->connect_error) {
13     die("Connection failed: " . $conn->connect_error);
14 }
15
```

1. Connexion à la base de données :

- Les variables \$servername, \$username, \$password, et \$dbname contiennent les informations nécessaires pour se connecter à la base de données MySQL.
- new mysqli(\$servername, \$username, \$password, \$dbname) crée une connexion à la base de données.
- if (\$conn->connect_error) vérifie si la connexion a échoué et affiche un message d'erreur en utilisant die().

```
15 // Initialiser les variables pour la Latitude et la Longitude
16 $latitude = null;
17 $longitude = null;
18 $tauxgaz = null;
19 $temperature = null;
20
21
```

1. Initialisation des variables :

- Les variables \$latitude, \$longitude, \$tauxgaz, et \$temperature sont initialisées à null.

```
22 // Requête SQL pour récupérer la dernière latitude non nulle
23 $sql = "SELECT latitude, longitude FROM ouvrier1 WHERE latitude IS NOT NULL ORDER BY id DESC LIMIT 1";
24 $result = $conn->query($sql);
25
26 // Vérifier si une ligne est trouvée
27 if ($result->num_rows > 0) {
28     $row = $result->fetch_assoc();
29     $latitude = $row["latitude"];
30     $longitude = $row["longitude"];
31 } else {
32     echo "0 results";
33 }
```

1. Requête SQL pour la latitude et la longitude :

- \$sql contient une requête SQL pour sélectionner les dernières valeurs non nulles de latitude et longitude dans la table ouvrier1.
- \$result = \$conn->query(\$sql) exécute la requête.
- if (\$result->num_rows > 0) vérifie si des résultats sont trouvés. Si oui, les valeurs sont récupérées et assignées aux variables \$latitude et \$longitude.

```

34
35 // Requête SQL pour récupérer la dernière ligne avec taux de gaz et température non nuls
36 $sql2 = "SELECT tauxgaz, température FROM ouvrier1 ORDER BY id DESC LIMIT 1";
37 $result2 = $conn->query($sql2);
38
39 // Vérifier si une ligne est trouvée
40 if ($result2->num_rows > 0) {
41     $row2 = $result2->fetch_assoc();
42     $tauxgaz = $row2["tauxgaz"];
43     $température = $row2["température"];
44 } else {
45     echo "0 results";
46 }
47

```

1. Requête SQL pour le taux de gaz et la température :

- \$sql2 contient une requête SQL pour sélectionner les dernières valeurs de tauxgaz et température dans la table ouvrier1.
- \$result2 = \$conn->query(\$sql2) exécute la requête.
- if (\$result2->num_rows > 0) vérifie si des résultats sont trouvés. Si oui, les valeurs sont récupérées et assignées aux variables \$tauxgaz et \$température.

```

47
48 // Fermer la connexion à la base de données
49 $conn->close();
50 ?>
51

```

1. Fermeture de la connexion :

- conn->close() ferme la connexion à la base de données.

code html et javascript de la page du casque n°1 :

```

69
70     <div id="map" style="width: 600px; height: 400px;"></div>
71
72     <script>
73         var map = L.map('map');
74         L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
75             attribution: '@ OpenStreetMap contributors'
76         }).addTo(map);
77
78         // Utiliser les valeurs de latitude et de longitude récupérées depuis la base de données
79         var latitude = <?php echo $latitude !== null ? $latitude : '0'; ?>; // Latitude
80         var longitude = <?php echo $longitude !== null ? $longitude : '0'; ?>; // Longitude
81
82         // Ajuster automatiquement le niveau de zoom pour que la carte soit centrée sur le marqueur
83         map.setView([latitude, longitude], 12);
84
85         // Ajouter un marqueur à la position spécifiée
86         var marker = L.marker([latitude, longitude]).addTo(map)
87             .bindPopup('Position GPS');
88     </script>
89
90     <div class="gaz">
91         capteur de gaz :
92         <?php
93             if ($tauxgaz >= 300) {
94                 echo "<div class='gazd'>Attention GAZ présent!!!</div>";

```

```

95     } else {
96         echo "Pas de gaz présent.";
97     }
98 ?>
99 </div>
100
101 <br>
102
103 <div class="temp">
104     capteur de température :
105     <?php
106     if ($temperature >= 50) {
107         echo "<div class='tempd'>Attention forte température!! : " . $temperature . "°C</div>";
108     } else {
109         echo "Température normale : " . $temperature . "°C";
110     }
111 ?>
112 </div>
113
114 </body>
115
116 </html>
117

```

1. Carte Leaflet :

- <div id="map" style="width: 600px; height: 400px;"></div> définit un conteneur pour la carte avec des dimensions spécifiques.
- Le script JavaScript initialise la carte, configure la couche de tuiles OpenStreetMap et ajoute un marqueur à la position spécifiée par \$latitude et \$longitude.

2. Affichage des capteurs :

- Les sections suivantes affichent les valeurs des capteurs de gaz et de température. Si le taux de gaz dépasse 300, un avertissement est affiché. De même, si la température dépasse 50°C, un avertissement est affiché.

code php de la page d'historique d'évènement :

```

1 <?php
2 include "C:/wamp64/www/gps/templates/header.php";
3 ?>
4

```

1. Inclusion de l'en-tête :

- La ligne include "C:/wamp64/www/gps/templates/header.php"; inclut le fichier PHP spécifié pour insérer le contenu de l'en-tête dans cette page.

```

4 <!DOCTYPE html>
5 <html lang="en">
6 <head>
7     <meta charset="UTF-8">
8     <meta name="viewport" content="width=device-width, initial-scale=1.0">
9     <title>Document</title>
10
11 </head>
12 <body>
13 </br>
14
15         <form method="post" action="recherche.php">
16             <input type="text" name="monBouton" placeholder="AAAA-MM-JJ"/>
17             <input type="submit" value="Rechercher">
18         </form>
19
20 </body>
21 </html>

```

1. Formulaire de recherche :

- Le formulaire permet de rechercher des informations par date. Il envoie une requête POST au fichier recherche.php.

```
24  
25 $servername = "localhost"; // Adresse du serveur MySQL  
26 $username = "root"; // Nom d'utilisateur MySQL  
27 $password = ""; // Mot de passe MySQL  
28 $dbname = "casque"; // Nom de la base de données MySQL  
29  
30 // Créer une connexion  
31 $conn = new mysqli($servername, $username, $password, $dbname);  
32  
33 // Vérifier la connexion  
34 if ($conn->connect_error) {  
35 die("Connection failed: " . $conn->connect_error);  
36 }  
37
```

1. Connexion à la base de données :

- Les variables \$servername, \$username, \$password, et \$dbname sont utilisées pour configurer la connexion à la base de données MySQL.
- new mysqli(\$servername, \$username, \$password, \$dbname) crée une nouvelle connexion MySQL.
- if (\$conn->connect_error) vérifie s'il y a une erreur de connexion et arrête l'exécution avec un message d'erreur.

```
38 // Initialiser les variables pour la latitude et la longitude  
39 $latitude = null;  
40 $longitude = null;  
41  
42 // Requête SQL pour récupérer la dernière Latitude non nulle  
43 $sql = "SELECT latitude, longitude FROM ouvrier1 WHERE latitude IS NOT NULL ORDER BY id DESC LIMIT 1";  
44 $result = $conn->query($sql);  
45  
46 // Vérifier si une ligne est trouvée  
47 if ($result->num_rows > 0) {  
  
48     $row = $result->fetch_assoc();  
49     $latitude = $row["latitude"];  
50     $longitude = $row["longitude"];  
51 }  
52
```

1. Récupération de la latitude et de la longitude :

- SELECT latitude, longitude FROM ouvrier1 WHERE latitude IS NOT NULL ORDER BY id DESC LIMIT 1 récupère la dernière latitude et longitude non nulles de la table ouvrier1.
- Les valeurs sont ensuite assignées aux variables \$latitude et \$longitude.

```
52  
53 // Requête SQL pour sélectionner la date correspondant à l'ID le plus élevé pour chaque ouvrier  
54 $sql1 = "SELECT tauxgaz, date, temperature FROM ouvrier1";  
55 $result1 = $conn->query($sql1);  
56  
57 if ($result1->num_rows > 0) {  
    // Output data of each row  
    foreach ($result1 as $row) {  
        if ($row["tauxgaz"] >= 300 || $row["temperature"] >= 70) {  
            echo "gaz ou haute température présent: Taux de Gaz : " . $row["tauxgaz"] . " | température: " . $row["temperature"] . " - Date: " . $row["date"] . "<br>";  
        }  
    }  
} else {  
    echo "Pas d'information inquiétante";  
}  
67
```

1. Récupération et affichage des données des capteurs :

- SELECT tauxgaz, date, temperature FROM ouvrier1 récupère le taux de gaz, la date et la température de la table ouvrier1.
- if (\$row["tauxgaz"] >= 300 || \$row["temperature"] >= 70) vérifie si le taux de gaz dépasse 300 ou si la température dépasse 70. Si c'est le cas, un message d'avertissement est affiché.

code pour la barre de recherche :

```
2  <?php
3  $servername = "localhost";
4  $username = "root";
5  $password = "";
6  $dbname = "casque";
7
8  include "C:\wamp64\www\gps\templates\header.php";
9
10 // Créer une connexion
11 $conn = new mysqli($servername, $username, $password, $dbname);
12
13 // Vérifier la connexion
14 if ($conn->connect_error) {
15     die("Connection failed: " . $conn->connect_error);
16 }
17
```

1. Connexion à la base de données :

- Les informations de connexion (serveur, utilisateur, mot de passe, base de données) sont utilisées pour établir une connexion à la base de données MySQL.
- new mysqli(\$servername, \$username, \$password, \$dbname) crée une connexion MySQL.
- if (\$conn->connect_error) vérifie s'il y a une erreur de connexion et arrête l'exécution avec un message d'erreur si c'est le cas.

```
17
18  if ($_SERVER["REQUEST_METHOD"] == "POST") {
19      if (isset($_POST['Recherche'])) {
20          echo "La recherche a été effectuée avec succès !";
21      }
22  }
```

1. Formulaire de recherche :

- Le formulaire HTML permet à l'utilisateur d'entrer une date pour rechercher des enregistrements dans la base de données.
- method="post" spécifie que les données du formulaire sont envoyées via la méthode POST.

```

18 if ($_SERVER["REQUEST_METHOD"] == "POST") {
19     if (isset($_POST['monBouton'])) {
20         // Vérifier l'état de la connexion avant d'utiliser real_escape_string
21         if ($conn->connect_errno) {
22             die("Erreur de connexion : " . $conn->connect_error);
23         }
24
25         $valeurBouton = $conn->real_escape_string($_POST['monBouton']);
26
27         // Vérifier l'état de la connexion avant d'exécuter la requête SQL
28         if ($conn->connect_errno) {
29             die("Erreur de connexion : " . $conn->connect_error);
30         }
31
32         $sql = "SELECT * FROM ouvrier1 WHERE date LIKE '%$valeurBouton%'";
33
34         $result = $conn->query($sql);
35
36         if ($result === false) {
37             die("Erreur lors de l'exécution de la requête : " . $conn->error);
38         }
39
40         $found = false;
41
42         if ($result->num_rows > 0) {
43             while ($row = $result->fetch_assoc()) {
44                 $tauxgaz = $row["tauxgaz"];
45                 $temperature = $row["temperature"];
46                 if ($tauxgaz >= 300 || $temperature >= 70) {
47                     echo "Gaz ou haute température présent : Taux de Gaz : " . $tauxgaz . " | Température : " . $temperature . " - Date : " . $row["date"] . "<br>";
48                     $found = true;

```

```

48                         $found = true;
49                     }
50                 }
51             }
52
53             if (!$found) {
54                 echo "Pas d'information trouvée";
55             }
56         }
57     }

```

1. Traitement du formulaire en PHP :

- if (\$_SERVER["REQUEST_METHOD"] == "POST") vérifie si le formulaire a été soumis.
- if (isset(\$_POST['monBouton'])) vérifie si le champ du formulaire est défini.
- \$valeurBouton = \$conn->real_escape_string(\$_POST['monBouton']); sécurise la valeur entrée par l'utilisateur pour éviter les injections SQL.
- La requête SQL SELECT * FROM ouvrier1 WHERE date LIKE '%\$valeurBouton%' recherche les enregistrements dont la date correspond à la valeur entrée par l'utilisateur.

Les résultats de la requête sont vérifiés et affichés. Si le taux de gaz est supérieur ou égal à 300 ou si la température est supérieure ou égale à 70, un message d'alerte est affiché.

```
58  
59 // Fermer la connexion  
60 $conn->close();  
61 ?>
```

1. Fermeture de la connexion :

- conn->close() ferme la connexion à la base de données après avoir exécuté toutes les opérations nécessaire.

Conclusion

Au terme de ces quatre mois de projet, nous avons réussi à accomplir la plupart des tâches qui nous étaient assignées. Bien que nous ayons rencontré des difficultés techniques avec les shields, nous avons su faire preuve de flexibilité et d'ingéniosité pour contourner ces obstacles et mener à bien notre projet de casque connecté.

Les capteurs que nous avons intégrés sont désormais pleinement fonctionnels et communiquent efficacement avec le serveur, démontrant ainsi la viabilité technique de notre solution. Cette réussite technique est le fruit de notre persévérance et de notre capacité à résoudre les problèmes de manière créative et collaborative.

Ce projet nous a offert une opportunité précieuse d'apprendre à travailler en équipe sur un même projet complexe. Nous avons suivi un cahier des charges précis et avons respecté les délais impartis, ce qui nous a permis de développer des compétences en gestion de projet et en communication. Chacun d'entre nous a contribué au succès global, en apportant ses compétences uniques et en collaborant étroitement avec les autres membres de l'équipe.

En outre, cette expérience nous a permis de mieux comprendre l'importance de la planification, de la répartition des tâches et de la coordination au sein d'une équipe. Nous avons appris à anticiper les défis, à gérer les imprévus et à rester concentrés sur nos objectifs communs.

En conclusion, ce projet a été une expérience enrichissante qui nous a permis de développer nos compétences techniques et interpersonnelles. Il nous a préparés à affronter de futurs défis professionnels avec confiance et a renforcé notre capacité à travailler efficacement en équipe. Nous sommes fiers des progrès réalisés et de la solution que nous avons développée, et nous sommes impatients d'appliquer les leçons apprises dans nos futures entreprises.

Annexe

Etudiant 1 :

Annexe 1 E1: code entier du capteur de lumière:

```
1 import smbus
2 import time
3 import RPi.GPIO as GPIO
4
5 # Définition des adresses des capteur BH1750
6 BH1750_ADDR = 0x23
7 # Mode de mesure continu de 1 lux
8 CONTINUOUS_HIGH_RES_MODE = 0x10
9
10 # Initialisation du bus I2C
11 bus = smbus.SMBus(1)
12
13 # Configuration des pins GPIO
14 GPIO.setmode(GPIO.BOARD)
15 GPIO.setup(16, GPIO.OUT)
16
17 # initialisation de pwm_led à None
18 pwm_led = None
19
20 # Fonction pour lire la luminosité du capteur
21 def read_light():
22     data = bus.read_i2c_block_data(BH1750_ADDR, CONTINUOUS_HIGH_RES_MODE, 2)
23     light_level = int.from_bytes(data, byteorder='big')
24     return light_level
25
26 # Boucle principale
27 try:
28     while True:
29         light_level = read_light()
30         # Inverser le calcul de la luminosité de la LED en fonction du taux de luminosité
31         brightness = max(100 - int(light_level/2), 0) # Inverser le calcul de la luminosité
32
33         # Vérifier si un objet PWM existe déjà
34         if pwm_led:
35             pwm_led.stop() # Arrêter l'objet PWM existant
36             pwm_led = None # Réinitialiser l'objet PWM existant
37
38         # Créer un nouvel objet PWM
39         pwm_led = GPIO.PWM(16, 100)
40         pwm_led.start(brightness) # Démarrer avec la nouvelle luminosité
41
42         time.sleep(1)
43
44 # Arrêt du programme sur CTRL+C
45 except KeyboardInterrupt:
46     pass
47
48 # Réinitialisation des pins GPIO
49 finally:
50     if pwm_led:
51         pwm_led.stop() # Arrêter l'objet PWM s'il existe encore
52     GPIO.cleanup()
```

Annexe 2 E1 :

code complet de l'accéléromètre:

```
import smbus
from time import sleep

# select the correct i2c bus for this revision of Raspberry Pi
revision = ([l[12:-1] for l in open('/proc/cpuinfo','r').readlines() if l[:8]== "Revision"]+['0000'][0])
bus = smbus.SMBus(1 if int(revision, 16) >= 4 else 0)

# ADXL345 constants
EARTH_GRAVITY_MS2    = 9.80665
SCALE_MULTIPLIER      = 0.004

DATA_FORMAT           = 0x31
BW_RATE               = 0x2C
POWER_CTL             = 0x2D

BW_RATE_1600HZ        = 0x0F
BW_RATE_800HZ         = 0x0E
BW_RATE_400HZ         = 0x0D
BW_RATE_200HZ         = 0x0C
BW_RATE_100HZ         = 0x0B
BW_RATE_50HZ          = 0x0A
BW_RATE_25HZ          = 0x09

RANGE_2G              = 0x00
RANGE_4G              = 0x01
RANGE_8G              = 0x02
RANGE_16G             = 0x03

MEASURE                = 0x08
AXES_DATA              = 0x32

class ADXL345:

    address = None

    def __init__(self, address = 0x53):
        self.address = address
        self.setBandwidthRate(BW_RATE_100HZ)
        self.setRange(RANGE_2G)
        self.enableMeasurement()

    def enableMeasurement(self):
        bus.write_byte_data(self.address, POWER_CTL, MEASURE)

    def setBandwidthRate(self, rate_flag):
        bus.write_byte_data(self.address, BW_RATE, rate_flag)

    # set the measurement range for 10-bit readings
```

```

def setRange(self, range_flag):
    value = bus.read_byte_data(self.address, DATA_FORMAT)

    value &= ~0x0F;
    value |= range_flag;
    value |= 0x08;

    bus.write_byte_data(self.address, DATA_FORMAT, value)

# returns the current reading from the sensor for each axis
#
# parameter gforce:
#   False (default): result is returned in m/s^2
#   True           : result is returned in gs
def getAxes(self, gforce = False):
    bytes = bus.read_i2c_block_data(self.address, AXES_DATA, 6)

    x = bytes[0] | (bytes[1] << 8)
    if(x & (1 << 16 - 1)):
        x = x - (1<<16)

    y = bytes[2] | (bytes[3] << 8)
    if(y & (1 << 16 - 1)):
        y = y - (1<<16)

    z = bytes[4] | (bytes[5] << 8)
    if(z & (1 << 16 - 1)):
        z = z - (1<<16)

    x = x * SCALE_MULTIPLIER
    y = y * SCALE_MULTIPLIER
    z = z * SCALE_MULTIPLIER

    if gforce == False:
        x = x * EARTH_GRAVITY_MS2
        y = y * EARTH_GRAVITY_MS2
        z = z * EARTH_GRAVITY_MS2

    x = round(x, 4)
    y = round(y, 4)
    z = round(z, 4)

    return {"x": x, "y": y, "z": z}

if __name__ == "__main__":
    adxl345 = ADXL345()

```

```

while True:
    axes = adxl345.getAxes(True)
    print ("ADXL345 on address 0x%x:" % (adxl345.address))
    print ("    x = %.3fG" % ( axes['x'] ))
    print ("    y = %.3fG" % ( axes['y'] ))
    print ("    z = %.3fG" % ( axes['z'] ))

    # Attendre une seconde avant de récupérer et d'afficher de nouvelles données
    sleep(2)

```

Annexe 3 E1 :

code complet du bouton poussoir + l'accéléromètre:

```
1 import smbus
2 from time import sleep, time
3 import RPi.GPIO as GPIO
4
5 # Configuration des GPIO
6 BUTTON_PIN = 17 # Numéro de la broche GPIO pour le bouton
7 RESPONSE_TIMEOUT = 10 # Délai d'attente pour la réponse en secondes
8
9 # Définition des constantes pour l'accéléromètre
10 EARTH_GRAVITY_MS2 = 9.80665
11 SCALE_MULTIPLIER = 0.004
12
13 # Constantes pour les plages acceptables d'accélération
14 ACCEPTABLE_RANGE_MIN = -2 # Valeur minimale acceptable
15 ACCEPTABLE_RANGE_MAX = 2 # Valeur maximale acceptable
16
17 GPIO.setmode(GPIO.BCM)
18 GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
19
20 # Initialisation de l'accéléromètre
21 bus = smbus.SMBus(1)
22 adxl345_address = 0x53 # Adresse de l'accéléromètre
23 adxl345 = ADXL345(adxl345_address)
24
25 def check_button_response():
26     print("Est-ce que tout va bien ? Appuyez sur le bouton pour confirmer.")
27
28     start_time = time()
29     button_pressed = False
30
31     while time() - start_time < RESPONSE_TIMEOUT:
32         if GPIO.input(BUTTON_PIN) == GPIO.LOW: # Le bouton est appuyé
33             button_pressed = True
34             break
35         sleep(0.1) # Pause de 100ms pour éviter de vérifier trop fréquemment
36
37     if button_pressed:
38         print("Bouton appuyé, tout va bien.")
39     else:
40         print("Alerte : Pas de réponse du bouton !")
41
42 def check_acceleration():
43     axes = adxl345.getAxes(True)
44     x_acceleration = axes['x']
45     y_acceleration = axes['y']
46     z_acceleration = axes['z']
47
```

```
48     # Vérifier si les valeurs d'accélération sont en dehors de la plage acceptable
49     if (x_acceleration < ACCEPTABLE_RANGE_MIN or x_acceleration > ACCEPTABLE_RANGE_MAX or
50         y_acceleration < ACCEPTABLE_RANGE_MIN or y_acceleration > ACCEPTABLE_RANGE_MAX or
51         z_acceleration < ACCEPTABLE_RANGE_MIN or z_acceleration > ACCEPTABLE_RANGE_MAX):
52         print("Alerte : Problème avec les coordonnées !")
53         # Code pour envoyer un message d'alerte ici
54
55     try:
56         while True:
57             check_button_response()
58             check_acceleration()
59             sleep(30)  # Vérifie toutes les 30 secondes
60
61     except KeyboardInterrupt:
62         print("Programme arrêté par l'utilisateur")
63
64     finally:
65         GPIO.cleanup()
```

Etudiant 2 :

Annexe 1 E2 :

```
C:\> Users > kylli > Desktop > gazfinal2.py
 1  import RPi.GPIO as GPIO
 2  import time
 3  import mysql.connector
 4  import Adafruit_DHT
 5
 6  # Initialisation des broches GPIO
 7  GPIO.setmode(GPIO.BCM)
 8  CLK = 11
 9  DOUT = 9
10  DIN = 10
11  CS = 8
12  GPIO.setup(CLK, GPIO.OUT)
13  GPIO.setup(DOUT, GPIO.IN)
14  GPIO.setup(DIN, GPIO.OUT)
15  GPIO.setup(CS, GPIO.OUT)
16
17  # Connexion à la base de données Mariadb localhost
18  conn_local = mysql.connector.connect(
19      host="localhost",
20      user="root",
21      password="pi",
22      database="casque",
23      auth_plugin='mysql_native_password'
24  )
25  cursor_local = conn_local.cursor()
26
27  # Connexion à la base de données MySQL phpmyadmin
28  conn_distant = mysql.connector.connect(
29      host="172.16.112.193",
30      user="root",
31      password="",
32      database="casque",
33      auth_plugin='mysql_native_password'
34  )
35  cursor_distant = conn_distant.cursor()
36
37  gas_sensor = 6 # Canal du capteur de gaz
38  sensor = Adafruit_DHT.DHT11
39  pin = 4
40
41  def readadc(num, clk, dout, din, cs):
42      if ((num > 7) or (num < 0)):
43          return -1
44
45      GPIO.output(cs, 1)
46      GPIO.output(clk, 0)
47      GPIO.output(cs, 0)
```

```

48     command = num
49     command |= 0x18
50     command <= 3
51
52     for i in range(5):
53         if (command & 0x80):
54             GPIO.output(din, 1)
55         else:
56             GPIO.output(din, 0)
57
58         command <= 1
59         GPIO.output(clk, 1)
60         GPIO.output(clk, 0)
61
62     out = 0
63
64     for i in range(12):
65         GPIO.output(clk, 1)
66         GPIO.output(clk, 0)
67         out <= 1
68         out |= GPIO.input(dout)
69
70     GPIO.output(cs, 1)
71     out >>= 1
72     return out
73
74
75 try:
76     while True:
77         # Lecture du capteur de gaz
78         gas_value = readadc(gas_sensor, CLK, DOUT, DIN, CS)
79         print("Taux de gaz : {}".format(gas_value))
80
81         # Lecture du capteur de température et d'humidité
82         humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
83
84         if humidity is not None and temperature is not None:
85             print('Temp={0:0.1f}*C  Humidity={1:0.1f}%'.format(temperature, humidity))
86

```

```

87         # Insertion des données dans la base de données locale
88         sql_insert_local = "INSERT INTO ouvrier1 (temperature, humidite, tauxgaz) VALUES (%s, %s, %s)"
89         val_local = (temperature, humidity, gas_value)
90         cursor_local.execute(sql_insert_local, val_local)
91         conn_local.commit()
92         print("Données insérées dans la base de données locale.")
93
94         # Insertion des données dans la base de données distante
95         sql_insert_distante = "INSERT INTO ouvrier1 (temperature, humidite, tauxgaz) VALUES (%s, %s, %s)"
96         val_distante = (temperature, humidity, gas_value)
97         cursor_distante.execute(sql_insert_distante, val_distante)
98         conn_distante.commit()
99         print("Données insérées dans la base de données distante.")
100     else:
101         print('Échec de la lecture. Réessayer!')
102
103     time.sleep(5)
104 except KeyboardInterrupt:
105     GPIO.cleanup()
106     conn_local.close()
107     conn_distante.close()
108

```

Etudiant 3 :

Annexe 1 E3 :

code en Python pour la récupération, traitement et stockage des données du GPS

```
3 # Import des modules nécessaires
4 import serial # Module pour la communication série
5 import pynmea2 # Module pour le traitement des données NMEA
6 import mysql.connector # Module pour la connexion à la base de données MySQL/MariaDB
7 import time # Module pour la gestion du temps
8
9 tabnine: test | explain | document | ask
10 def connect_local_database():
11     # Fonction pour se connecter à la base de données locale MariaDB
12     return mysql.connector.connect(
13         host="localhost", # Adresse IP ou nom d'hôte du serveur MariaDB Local
14         user="root", # Nom d'utilisateur de la base de données MariaDB
15         password="pi", # Mot de passe de la base de données MariaDB
16         database="casque", # Nom de la base de données MariaDB
17         auth_plugin='mysql_native_password'
18     )
19
20 tabnine: test | explain | document | ask
21 def connect_remote_database():
22     # Fonction pour se connecter à la base de données distante MariaDB
23     return mysql.connector.connect(
24         host="172.16.112.193", # Adresse IP ou nom d'hôte du serveur MariaDB distant
25         user="root", # Nom d'utilisateur de la base de données MariaDB
26         password="", # Mot de passe de la base de données MariaDB
27         database="casque", # Nom de la base de données MariaDB
28         auth_plugin='mysql_native_password'
29     )
30
31 tabnine: test | explain | document | ask
32 def read_gps_data(db_connection, db_cursor):
33     # Fonction pour lire les données GPS
34     serial_port = serial.Serial("/dev/ttys0", 9600, timeout=0.5) # Configuration du port série
35
36     try:
37         while True:
38             start_time = time.time() # Temps de début de la boucle
39
40             raw_data = serial_port.readline().decode("utf-8").strip() # Lecture des données brutes du GPS
41             print("Raw Data:", raw_data) # Affichage de la trame NMEA sans les caractères de nouvelle ligne
42
43             try:
44                 nmea_data = pynmea2.parse(raw_data) # Analyse de la trame NMEA
45
46                 if isinstance(nmea_data, pynmea2.GGA): # Vérification du type de trame (GPGGA)
47                     if nmea_data.gps_qual == 1: # Vérification de la qualité du fix GPS
48                         latitude = nmea_data.latitude # Récupération de la latitude
49                         longitude = nmea_data.longitude # Récupération de la longitude
```

```

47
48         # Insertion des données dans la base de données
49         sql_insert_query = "INSERT INTO ouvrier1 (latitude, longitude) VALUES (%s, %s)"
50         insert_values = (latitude, longitude)
51         print("Insert Values:", insert_values) # Affichage des valeurs à insérer dans la base de données
52         db_cursor.execute(sql_insert_query, insert_values)
53         db_connection.commit()
54
55         print("Données insérées dans la base de données.")
56
57     except pynmea2.ParseError as e:
58         print("Erreur de parsing:", e)
59         continue # Passage à la prochaine itération de la boucle en cas d'erreur de parsing
60
61     elapsed_time = time.time() - start_time # Temps écoulé depuis le début de la boucle
62     time_to_sleep = 5 - elapsed_time # Calcul du temps restant à attendre
63     if time_to_sleep > 0:
64         time.sleep(time_to_sleep) # Attendre le temps restant pour atteindre 5 secondes
65
66 except KeyboardInterrupt:
67     print("Arrêt de la lecture GPS.")
68 except Exception as e:
69     print("Erreur:", e)
70
71 if __name__ == "__main__":
72     # Connexion à la base de données locale
73     local_db_connection = connect_local_database()
74     local_db_cursor = local_db_connection.cursor()
75
76     # Lecture des données GPS et stockage dans la base de données locale
77     read_gps_data(local_db_connection, local_db_cursor)
78
79     # Connexion à la base de données distante
80     remote_db_connection = connect_remote_database()
81     remote_db_cursor = remote_db_connection.cursor()
82
83     # Lecture des données GPS et stockage dans la base de données distante
84     read_gps_data(remote_db_connection, remote_db_cursor)
85

```

Annexe 2 E3 :

code pour la page du casque n°1:

```
1 <?php
2 // Connexion à la base de données
3 $servername = "localhost"; // Adresse du serveur MySQL
4 $username = "root"; // Nom d'utilisateur MySQL
5 $password = ""; // Mot de passe MySQL
6 $dbname = "casque"; // Nom de la base de données MySQL
7
8 // Créer une connexion
9 $conn = new mysqli($servername, $username, $password, $dbname);
10
11 // Vérifier la connexion
12 if ($conn->connect_error) {
13     die("Connection failed: " . $conn->connect_error);
14 }
15
16 // Initialiser les variables pour la latitude et la longitude
17 $latitude = null;
18 $longitude = null;
19 $tauxgaz = null;
20 $temperature = null;
21
22 // Requête SQL pour récupérer la dernière latitude non nulle
23 $sql = "SELECT latitude, longitude FROM ouvrier1 WHERE latitude IS NOT NULL ORDER BY id DESC LIMIT 1";
24 $result = $conn->query($sql);
25
26 // Vérifier si une ligne est trouvée
27 if ($result->num_rows > 0) {
28     $row = $result->fetch_assoc();
29     $latitude = $row["latitude"];
30     $longitude = $row["longitude"];
31 } else {
32     echo "0 results";
33 }
34
35 // Requête SQL pour récupérer la dernière ligne avec taux de gaz et température non nuls
36 $sql2 = "SELECT tauxgaz, temperature FROM ouvrier1 ORDER BY id DESC LIMIT 1";
37 $result2 = $conn->query($sql2);
38
39 // Vérifier si une ligne est trouvée
40 if ($result2->num_rows > 0) {
41     $row2 = $result2->fetch_assoc();
42     $tauxgaz = $row2["tauxgaz"];
43     $temperature = $row2["temperature"];
44 } else {
45     echo "0 results";
46 }
```

```

48 // Fermer la connexion à la base de données
49 $conn->close();
50 ?>
51
52 <!DOCTYPE html>
53 <html>
54
55 <head>
56     <title>Ouvrier N°1</title>
57     <link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" />
58     <script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
59     <link rel="stylesheet" href="index.css">
60 </head>
61
62 <body>
63
64     <?php include "C:/wamp64/www/gps/templates/header.php" ?>
65
66     <a href="index.php">
67         <h1>Ouvrier N°1</h1>
68     </a>
69
70     <div id="map" style="width: 600px; height: 400px;"></div>
71
72     <script>
73         var map = L.map('map');
74         L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}.png', {
75             attribution: '© OpenStreetMap contributors'
76         }).addTo(map);
77
78         // Utiliser les valeurs de Latitude et de Longitude récupérées depuis la base de données
79         var latitude = <?php echo $latitude !== null ? $latitude : '0'; ?>; // Latitude
80         var longitude = <?php echo $longitude !== null ? $longitude : '0'; ?>; // Longitude
81
82         // Ajuster automatiquement le niveau de zoom pour que la carte soit centrée sur le marqueur
83         map.setView([latitude, longitude], 12);
84
85         // Ajouter un marqueur à la position spécifiée
86         var marker = L.marker([latitude, longitude]).addTo(map)
87             .bindPopup('Position GPS');
88     </script>
89
90     <div class="gaz">
91         capteur de gaz :
92         <?php
93             if ($tauxgaz >= 300) {
94                 echo "<div class='gazd'>Attention GAZ présent!!!</div>";
95
96             } else {
97                 echo "Pas de gaz présent.";
98             }
99         </div>
100
101     <br>
102
103     <div class="temp">
104         capteur de température :
105         <?php
106             if ($temperature >= 50) {
107                 echo "<div class='tempd'>Attention forte température!! : " . $temperature . "°C</div>";
108             } else {
109                 echo "Température normale : " . $temperature . "°C";
110             }
111         </div>
112     </div>
113
114 </body>
115
116 </html>

```

Annexe 3 E3 :

code pour l'historique d'évènement :

```
1  <?php
2  include "C:\wamp64\www\gps\templates\header.php";
3  ?>
4
5  <!DOCTYPE html>
6  <html lang="en">
7  <head>
8      <meta charset="UTF-8">
9      <meta name="viewport" content="width=device-width, initial-scale=1.0">
10     <title>Document</title>
11 </head>
12 <body>
13 </br>
14
15         <form method="post" action="recherche.php">
16             <input type="text" name="monBouton" placeholder="AAAA-MM-JJ"/>
17             <input type="submit" value="Rechercher">
18         </form>
19 </body>
20 </html>
21 <h1><a href="notif.php">retour</a><br></h1>
22 <h2>
23 <?php
24
25 $servername = "localhost"; // Adresse du serveur MySQL
26 $username = "root"; // Nom d'utilisateur MySQL
27 $password = ""; // Mot de passe MySQL
28 $dbname = "casque"; // Nom de la base de données MySQL
29
30 // Créer une connexion
31 $conn = new mysqli($servername, $username, $password, $dbname);
32
33 // Vérifier la connexion
34 if ($conn->connect_error) {
35     die("Connection failed: " . $conn->connect_error);
36 }
37
38 // Initialiser les variables pour la latitude et la longitude
39 $latitude = null;
40 $longitude = null;
41
42 // Requête SQL pour récupérer la dernière latitude non nulle
43 $sql = "SELECT latitude, longitude FROM ouvrier WHERE latitude IS NOT NULL ORDER BY id DESC LIMIT 1";
44 $result = $conn->query($sql);
45
46 // Vérifier si une ligne est trouvée
47 if ($result->num_rows > 0) {
48     $row = $result->fetch_assoc();
49
50     $latitude = $row["latitude"];
51     $longitude = $row["longitude"];
52 }
53
54 // Requête SQL pour sélectionner la date correspondant à l'ID le plus élevé pour chaque ouvrier
55 $sql1 = "SELECT tauxgaz, date, temperature FROM ouvrier";
56 $result1 = $conn->query($sql1);
57
58 if ($result1->num_rows > 0) {
59     // Output data of each row
60     foreach ($result1 as $row) {
61         if ($row["tauxgaz"] >= 300 || $row["temperature"] >= 70) {
62             echo "gaz ou haute température présent: Taux de Gaz : " . $row["tauxgaz"] . " | température: " . $row["temperature"] . " - Date: " . $row["date"] . "<br>";
63         }
64     } else {
65         echo "Pas d'information inquiétante";
66     }
67
68
69
70 ?>
71 </h2>
72
```

Annexe 4 E3 :

code php pour la recherche :

```
1 <h2>
2 <?php
3 $servername = "localhost";
4 $username = "root";
5 $password = "";
6 $dbname = "casque";
7
8 include "C:\wamp64\www\gps\templates\header.php";
9
10 // Créer une connexion
11 $conn = new mysqli($servername, $username, $password, $dbname);
12
13 // Vérifier la connexion
14 if ($conn->connect_error) {
15     die("Connection failed: " . $conn->connect_error);
16 }
17
18 if ($_SERVER["REQUEST_METHOD"] == "POST") {
19     if (isset($_POST['monBouton'])) {
20         // Vérifier l'état de la connexion avant d'utiliser real_escape_string
21         if ($conn->connect_errno) {
22             die("Erreur de connexion : " . $conn->connect_error);
23         }
24
25         $valeurBouton = $conn->real_escape_string($_POST['monBouton']);
26
27         // Vérifier l'état de la connexion avant d'exécuter la requête SQL
28         if ($conn->connect_errno) {
29             die("Erreur de connexion : " . $conn->connect_error);
30         }
31
32         $sql = "SELECT * FROM ouvrier1 WHERE date LIKE '%$valeurBouton%'";
33
34         $result = $conn->query($sql);
35
36         if ($result === false) {
37             die("Erreur lors de l'exécution de la requête : " . $conn->error);
38         }
39
40         $found = false;
41
42         if ($result->num_rows > 0) {
43             while ($row = $result->fetch_assoc()) {
44                 $tauxgaz = $row["tauxgaz"];
45                 $temperature = $row["temperature"];
46                 if ($tauxgaz >= 300 || $temperature >= 70) {
47                     echo "Gaz ou haute température présent : Taux de Gaz : " . $tauxgaz . " | Température : " . $temperature . " - Date : " . $row["date"] . "<br>";
48                     $found = true;
49
50             }
51         }
52
53         if (!$found) {
54             echo "Pas d'information trouvée";
55         }
56     }
57 }
58
59 // Fermer la connexion
60 $conn->close();
61 ?>
62 </h2>
```