المدرسة الوطنية للعلوم التطبيقية - فاس
ⵜⵉⵎⵙⵙⵓⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵣⵎⵎⵓⵔⵉ ⵜⵉⵎⵓⵣⵓⵔⵉ
Ecole Nationale des Sciences Appliquées de Fès

جامعة سيدي محمد بن عبد الله بفاس
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴰⵎⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ ⴱⴼⴰⵙ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

Projet de fin d'année

3ème année cycle d'ingénieur
GÉNIE DES SYSTÈMES EMBARQUÉS ET INFORMATIQUE INDUSTRIELLE

Système IoT de Communication Véhicule-Feux de Circulation Basé sur MQTT

Soutenu le 04/02/2025

Préparé par : - Arab Insaf
- El Bouraqqady Oumaima

Encadré par : Pr. El Bdouri Abdelali

Jury :

- Pr. El Bdouri Abdelali
- Pr. Alami Marktani Malika

Année universitaire : 2024 – 2025

Remerciements

Nous tenons à exprimer notre profonde gratitude au professeur El Bdouri Abdelali pour son encadrement, ses conseils précieux et son soutien tout au long de ce projet. Son expertise et sa disponibilité ont été d'une aide inestimable dans la réalisation de ce travail.

Nous remercions également les membres du jury pour l'intérêt porté à notre projet et pour leurs remarques constructives qui contribueront à notre progression.

Nos sincères remerciements vont aussi à l'École Nationale des Sciences Appliquées de Fès, ainsi qu'à l'ensemble du corps professoral du département de Génie des Systèmes Embarqués et Informatique Industrielle pour la qualité de l'enseignement et de l'encadrement dont nous avons bénéficié tout au long de notre formation.

Enfin, nous adressons une pensée particulière à nos familles, amis et collègues pour leur soutien indéfectible et leurs encouragements constants.

Abstract

Real-time communication between vehicles and traffic infrastructure has become crucial with the growing need for smarter and safer transportation systems. The primary problematic addressed in this project is the lack of anticipation for drivers approaching traffic lights, which can lead to sudden braking, increased fuel consumption, and higher risks of accidents.

To overcome this issue, we propose an IoT-based system that enables real-time communication between traffic lights and vehicles. The project involves developing a prototype where a vehicle receives a notification through a Human-Machine Interface (HMI) application before reaching a traffic light, informing the driver about its current state (red, orange, or green). This system is built using a Raspberry Pi in both the traffic light and the vehicle, with MQTT (Message Queuing Telemetry Transport) over WiFi as the communication protocol for efficient and low-latency data exchange.

The HMI application provides clear and intuitive feedback, displaying real-time traffic light updates. Additionally, it includes dynamic guidance messages to assist drivers in making informed decisions based on the light's status. The solution contributes to better traffic flow, enhanced road safety, and improved driving comfort, demonstrating the potential of IoT in connected vehicles and smart city applications.

This project showcases how vehicle-to-infrastructure (V2I) communication can revolutionize urban mobility, making road systems more intelligent and responsive to real-world driving conditions.

Keys words : IoT (Internet of Things), Connected vehicle, Smart traffic light, V2I communication (Vehicle-to-Infrastructure), Raspberry Pi, MQTT (Message Queuing Telemetry Transport), Human-Machine Interface (HMI).

Résumé

Avec le besoin croissant de systèmes de transport plus intelligents et plus sûrs, la communication en temps réel entre les véhicules et les infrastructures routières devient essentielle. La problématique principale abordée dans ce projet est le manque d'anticipation des conducteurs à l'approche des feux de circulation, ce qui peut entraîner des freinages brusques, une surconsommation de carburant et un risque accru d'accidents.

Pour remédier à cette problématique, nous proposons un système basé sur l'IoT permettant une communication en temps réel entre les feux de circulation et les véhicules. Le projet consiste à développer un prototype où un véhicule reçoit une notification via une application d'Interface Homme-Machine (IHM) avant d'atteindre un feu tricolore, informant le conducteur de son état actuel (rouge, orange ou vert). Ce système repose sur l'utilisation d'un Raspberry Pi dans le feu de circulation et dans le véhicule, avec le protocole de communication MQTT (Message Queuing Telemetry Transport) sur WiFi pour assurer un échange de données efficace et à faible latence.

L'application IHM fournit un retour d'information clair et intuitif, affichant en temps réel l'état du feu de circulation. Elle intègre également des messages de guidage dynamiques pour aider les conducteurs à prendre des décisions éclairées en fonction du signal lumineux. Cette solution contribue à une meilleure fluidité du trafic, à une amélioration de la sécurité routière et à un plus grand confort de conduite, démontrant ainsi le potentiel de l'IoT dans les véhicules connectés et les applications de ville intelligente.

Ce projet illustre comment la communication véhicule-infrastructure (V2I) peut révolutionner la mobilité urbaine, rendant les systèmes routiers plus intelligents et plus réactifs aux conditions de conduite réelles.

Mots-clés : IoT (Internet des objets), Véhicule connecté, Feu de circulation intelligent, Communication V2I (Véhicule-à-Infrastructure), Raspberry Pi, MQTT (Message Queuing Telemetry Transport), Interface Homme-Machine (IHM).

ملخص

مع تزايد الحاجة إلى أنظمة نقل أكثر ذكاءً وأمانًا، أصبح الاتصال في الوقت الحقيقي بين المركبات والبنية التحتية المرورية أمرًا بالغ الأهمية. تتمثل الإشكالية الرئيسية التي يعالجها هذا المشروع في غياب التوقع لدى السائقين عند اقترابهم من إشارات المرور، مما قد يؤدي إلى فرملة مفاجئة، وزيادة استهلاك الوقود، وارتفاع مخاطر الحوادث.

لمعالجة هذه المشكلة، نقترح نظامًا يعتمد على إنترنت الأشياء يتيح الاتصال في الوقت الحقيقي بين إشارات المرور والمركبات. يتضمن المشروع تطوير نموذج أولي حيث تتلقى المركبة إشعارًا عبر تطبيق لواجهة الإنسان ـ الآلة قبل الوصول إلى إشارة المرور، لإعلام السائق بحالتها الحالية (أحمر، برتقالي، أو أخضر). يعتمد هذا النظام على استخدام الحاسوب المصغر في كل من إشارة المرور والمركبة، مع اعتماد بروتوكول نقل الرسائل عبر الشبكة اللاسلكية لضمان تبادل البيانات بكفاءة وزمن استجابة منخفض.

يوفر تطبيق واجهة الإنسان ـ الآلة ملاحظات واضحة وبديهية، حيث يعرض تحديثات إشارات المرور في الوقت الفعلي. بالإضافة إلى ذلك، يتضمن رسائل توجيه ديناميكية لمساعدة السائقين في اتخاذ قرارات جيدة بناءً على حالة الإشارة. تسهم هذه الحلول في تحسين انسيابية حركة المرور، وتعزيز السلامة على الطرق، وتحسين راحة القيادة، مما يبرز إمكانات إنترنت الأشياء في المركبات المتصلة وتطبيقات المدن الذكية.

يظهر هذا المشروع كيف يمكن لاتصال المركبة بالبنية التحتية إحداث ثورة في التنقل الحضري، مما يجعل أنظمة الطرق أكثر ذكاءً واستجابةً لظروف القيادة الواقعية.

الكلمات المفتاحية : إنترنت الأشياء، المركبة المتصلة، إشارة المرور الذكية، الاتصال (المركبة ـ البنية التحتية)، راسبيري باي، بروتوكول نقل رسائل قائمة الانتظار عن بعد، واجهة الإنسان ـ الآلة.

Table des matières

Table des figures

Liste d'abréviations

CSS Cascading Style Sheets

HTML HyperText Markup Language

IHM Interface Homme-Machine

IoT Internet of Things

MQTT Message Queuing Telemetry Transport

V2I Vehicle-to-Infrastructure

V2V Vehicle-to-Vehicle

Wi-Fi Wireless Fidelity

Introduction générale

Avec le développement des villes intelligentes et des véhicules connectés, l'optimisation des infrastructures de transport devient une priorité pour garantir une circulation fluide et sécurisée. L'un des principaux défis rencontrés est le manque d'anticipation des conducteurs face aux feux de signalisation, ce qui peut entraîner des freinages brusques, une augmentation des risques d'accident, une surconsommation de carburant et une perturbation du trafic.

Ce projet vise à répondre à cette problématique en mettant en place une solution IoT permettant une communication en temps réel entre les feux tricolores et les véhicules. Il repose sur l'utilisation de Raspberry Pi pour surveiller et transmettre l'état du feu via le protocole MQTT (Message Queuing Telemetry Transport) sur une connexion Wi-Fi. Un Raspberry Pi est installé au niveau du feu pour publier son état, tandis qu'un second Raspberry Pi embarqué dans le véhicule reçoit ces informations et les affiche sur une interface utilisateur intuitive. Cette approche permet aux conducteurs d'anticiper les changements de signalisation, réduisant ainsi les arrêts inutiles et contribuant à une meilleure fluidité du trafic.

Sur le plan technologique, le choix du protocole MQTT est particulièrement adapté aux systèmes IoT en raison de sa légèreté, de sa faible consommation de bande passante et de sa capacité à garantir des transmissions rapides et fiables. Grâce à ce système, le conducteur peut visualiser en temps réel l'état actuel du feu et prévoir son comportement en conséquence, améliorant ainsi la sécurité routière.

L'intégration de cette technologie s'inscrit dans la vision des transports intelligents, où la connectivité et l'automatisation jouent un rôle clé dans l'amélioration de la gestion du trafic urbain. En facilitant la communication véhicule-infrastructure (V2I), ce projet contribue non seulement à réduire les risques d'accidents et à optimiser les déplacements, mais aussi à diminuer la consommation énergétique et les émissions polluantes.

Ce rapport présente l'ensemble du travail réalisé et est structuré comme suit :

— Chapitre 1 : Présentation générale du projet.
— Chapitre 2 : Communication via le protocole MQTT.
— Chapitre 3 : Développement de l'Interface Homme-Machine (IHM).

Chapitre 1

Contexte générale du projet

1.1 Introduction

Ce chapitre a pour objectif de situer le projet dans son contexte général, à savoir la problématique qui a inspiré la création de notre système et la solution proposée.

1.2 Présentation générale du projet

1.2.1 Problématique

Dans un contexte de congestion croissante et d'urbanisation rapide, le manque d'anticipation des conducteurs face aux feux de circulation devient un problème majeur. En effet, l'incapacité à ajuster la vitesse en fonction de l'état des feux de signalisation entraîne des freinages brusques, perturbant la fluidité du trafic. Ces arrêts soudains augmentent non seulement la consommation de carburant en raison des accélérations et décélérations fréquentes, mais contribuent également à une détérioration de la qualité de l'air. De plus, le stress lié à ces changements de vitesse intempestifs peut engendrer des comportements imprévisibles et dangereux, augmentant le risque d'accidents.

Ce manque d'anticipation perturbe également la gestion du trafic, créant des embouteillages et des ralentissements inutiles, ce qui affecte négativement l'ensemble du réseau routier. En l'absence d'une communication en temps réel entre les véhicules et les infrastructures, les conducteurs ne peuvent pas prendre des décisions éclairées concernant leur conduite à l'approche d'un feu de signalisation. Cette situation engendre non seulement des pertes de temps mais impacte également la sécurité des usagers de la route, augmentant le risque d'incidents et de collisions.

1.2.2 Objectifs

L'objectif principal de ce projet est de répondre aux défis actuels du trafic urbain en s'appuyant sur des solutions technologiques innovantes. Pour ce faire, plusieurs objectifs spécifiques ont été définis :

— *Améliorer l'anticipation des conducteurs :* Offrir une information en temps réel sur l'état des feux de circulation (rouge, orange ou vert) afin que les conducteurs puissent ajuster leur conduite à l'avance, réduisant ainsi les freinages brusques et optimisant la fluidité du trafic.

— *Augmenter la sécurité routière :* En fournissant aux conducteurs des informations claires et à l'avance sur l'état des feux de circulation, le projet vise à réduire les risques d'accidents dus à des réactions imprévisibles ou des erreurs de jugement liées aux feux.
— *Démontrer le potentiel de l'IoT dans les systèmes de transport intelligents :* Ce projet vise à illustrer comment l'Internet des objets (IoT) peut transformer les infrastructures routières et rendre la mobilité urbaine plus connectée, réactive et durable.
— *Faciliter l'intégration d'infrastructures connectées :* Développer une architecture flexible et modulaire permettant une future évolution du système et l'intégration de nouvelles fonctionnalités ou de nouveaux appareils dans le cadre des villes intelligentes.

1.2.3 Solution proposée

Pour y remédier, nous proposons une solution basée sur l'IoT, permettant une communication en temps réel entre les feux de circulation et les véhicules. Le système alerte les conducteurs sur l'état du feu (rouge, orange ou vert) avant l'intersection via une Interface Homme-Machine (IHM) installée dans le véhicule. Ce système repose sur l'utilisation de Raspberry Pi tant au niveau des feux de circulation que des véhicules, avec le protocole MQTT (Message Queuing Telemetry Transport) sur WiFi pour assurer un échange de données efficace et à faible latence.
Ce projet vise à améliorer la fluidité du trafic, renforcer la sécurité routière et réduire la consommation de carburant en fournissant aux conducteurs des informations en temps réel, facilitant ainsi la prise de décisions. Il illustre le potentiel de l'IoT dans la communication véhicule-infrastructure et dans la création de systèmes de mobilité urbaine plus intelligents et réactifs.

1.2.4 Méthodologie

Le principe de la méthodologie SCRUM est de développer un logiciel de manière incrémentale en maintenant une liste totalement transparente des demandes d'évolutions ou de corrections à implémenter. Avec des livraisons très fréquentes, toutes les 4 semaines en moyenne, le client reçoit un logiciel fonctionnel à chaque itération. Plus nous avançons dans le projet, plus le logiciel est complet et possède toujours de plus en plus de fonctionnalités. Pour cela, la méthode s'appuie sur des développements itératifs à un rythme constant d'une durée de 2 à 4 semaines. [1]
La figure ci-dessous illustre le cycle de vie de la méthode SCRUM :

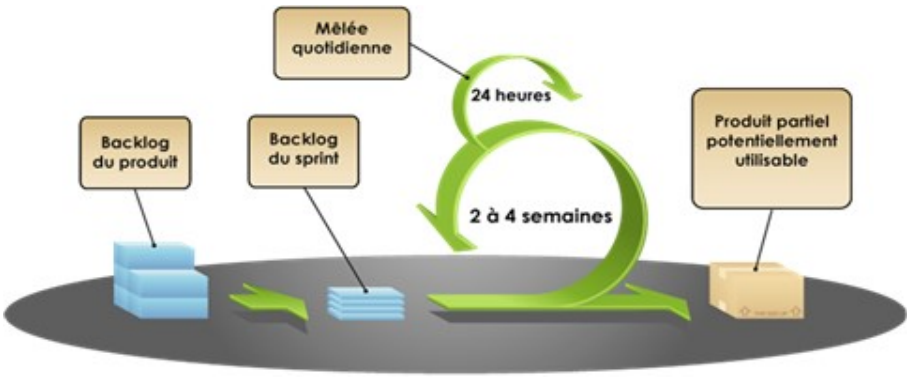

FIGURE 1.2.1 – Cycle de vie de la méthode SCRUM. [1]

Comme nous pouvons le remarquer dans cette figure, pour mettre en place la méthode SCRUM, il faut tout d'abord définir les différentes fonctionnalités de notre application qui forment le backlog du produit. Ensuite, nous procédons à la planification du sprint pour définir le plan détaillé d'une itération. Les sprints durent généralement deux à quatre semaines.[2]

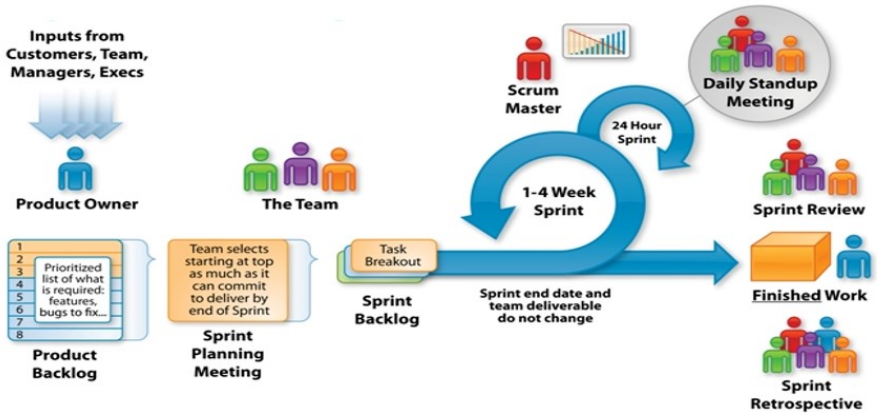

FIGURE 1.2.2 – Processus SCRUM.[2]

Durant un sprint, il y a toujours des réunions quotidiennes entre les différents collaborateurs du projet afin de présenter l'état d'avancement des différentes tâches en cours, les difficultés rencontrées ainsi que les tâches restantes à réaliser. Une fois le produit partiel est prêt, nous vérifions la conformité de ce qui a été fait durant le sprint et nous pouvons alors l'améliorer en procédant à l'étape de rétrospective.
Pour assurer une gestion efficace du projet, nous avons structuré notre travail en quatre grandes étapes, illustrées dans le diagramme de Gantt ci-dessous, permettant une planification claire des tâches et une répartition optimale du temps.

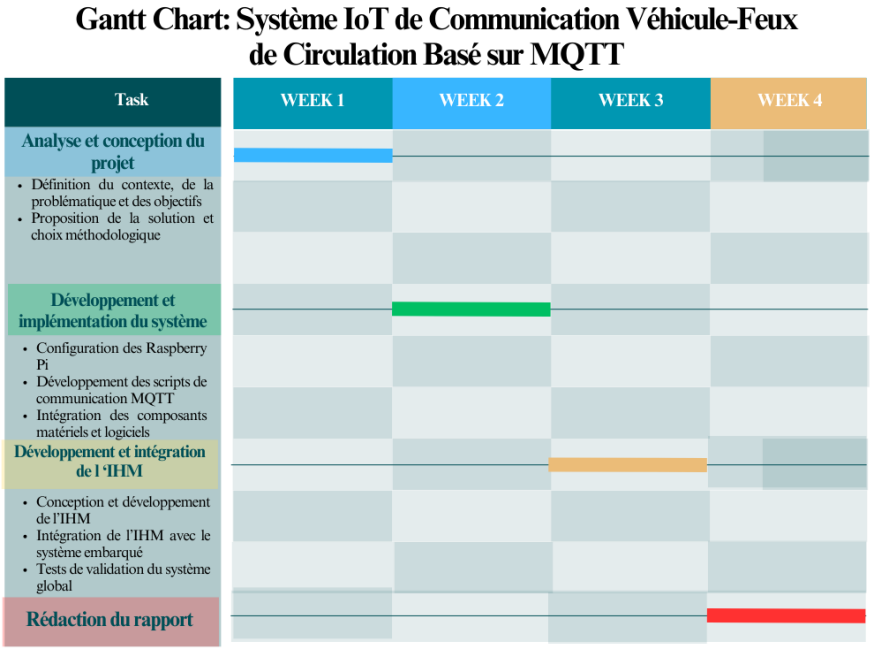

FIGURE 1.2.3 – Diagramme de GANTT pour la planification du projet.

1.3 Conclusion

Tout au long de ce chapitre, nous avons mis notre projet dans son cadre général, à savoir la présentation du projet et ses objectifs. Et pour mettre en place notre projet, on doit effectuer une étude préliminaire fonctionnelle et technique des besoins, cela fait l'objet du deuxième chapitre.

Chapitre 2

Communication via le protocole MQTT

2.1 Introduction

Ce chapitre décrit la mise en œuvre technique du projet, en détaillant les différentes étapes de développement et d'intégration des composants matériels et logiciels. La communication entre les feux de signalisation et le véhicule repose sur le protocole MQTT, assurant un échange de données rapide et efficace via un réseau Wi-Fi.

2.2 MQTT

2.2.1 Définition

MQTT, pour "Message Queuing Telemetry Transport", est un protocole open source de messagerie qui assure des communications non permanentes entre des appareils par le transport de leurs messages.
Il a été créé en 1999 par Andy Stanford-Clark, ingénieur chez IBM, et Arlen Nipper, chez EuroTech, principalement dans la communication M2M pour permettre à deux appareils utilisant des technologies différentes de communiquer. "Devenu une norme ISO en 2016, MQTT connectait déjà à cette date des millions d'appareils dans le monde entier, dans toutes sortes d'applications et d'industries. C'est une technologie d'avenir", affirme Fabien Pereira Vaz, technical sales manager chez Paessler AG. Les géants du web parmi lesquels AWS ou Microsoft utilisent MQTT pour remonter les données sur leur plateforme cloud. [3]

2.2.2 Architecture

MQTT est un protocole de messagerie push-subscribe basé sur le protocole TCP/IP. Dans l'architecture MQTT, il existe deux types de systèmes : les clients et les brokers (courtiers). Le broker est le serveur avec lequel les clients communiquent. Il reçoit les communications qui émanent des clients et les retransmet à d'autres clients. Ainsi, les clients ne communiquent pas directement entre eux, mais toujours par l'intermédiaire du broker. Chaque client peut être soit éditeur, soit abonné, soit les deux.
MQTT est un protocole orienté événements. Afin de minimiser le nombre de transmissions, les données ne sont envoyées ni à intervalles définis, ni en continu. Un client publie uniquement quand il a des informations à transmettre, et un broker n'envoie des informations aux abonnés que quand il reçoit de nouvelles données. [4]

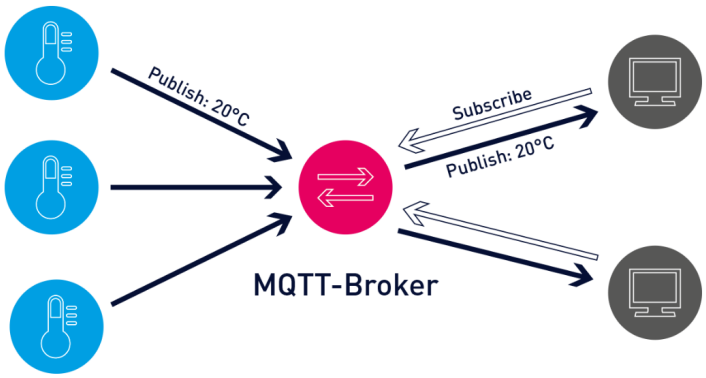

FIGURE 2.2.1 – Architecture MQTT. [4]

2.2.3 Avantages

Le protocole MQTT est devenu une norme pour la transmission de données IoT, car il offre les avantages suivants :

— *Léger et efficace :* Nécessite peu de ressources, adapté aux microcontrôleurs avec une faible consommation de bande passante.
— *Évolutif :* Supporte la connexion de millions d'appareils avec un code minimal et une faible consommation énergétique.
— *Fiable :* Conçu pour les réseaux instables avec trois niveaux de qualité de service garantissant la transmission des données.
— *Sécurisé :* Intègre le chiffrement et l'authentification via des protocoles modernes comme TLS1.3 et OAuth.
— *Bonne prise en charge :* Compatible avec plusieurs langages comme Python, facilitant son intégration rapide. [5]

2.3 Prototype du projet

L'architecture du projet repose sur une communication **véhicule-feu tricolore** via le protocole **MQTT** sur un réseau **Wi-Fi**. Un Raspberry Pi installé au feu surveille son état en temps réel (rouge, orange, vert) et publie ces informations sur un broker MQTT. Un second Raspberry Pi, embarqué dans le véhicule, souscrit à ces données et les transmet à une interface utilisateur développée avec **Flask** et **Flask-SocketIO**. Cette interface affiche l'état actuel du feu et son historique, permettant aux conducteurs d'anticiper les changements de signalisation et d'améliorer la fluidité du trafic.

2.3.1 Matériel utilisé

— 2 Raspberry Pi 4 avec un OS installé.
— LEDs (rouge, orange, verte) et résistances (220 Ω).
— Câbles de connexion.
— Une breadboard.
— Un point d'accès Wi-Fi (téléphone portable).

2.3.2 Circuit

Le circuit repose sur un Raspberry Pi connecté à un feu tricolore simulé à l'aide de trois LEDs (rouge, orange et verte), contrôlées via ses GPIOs. Les connexions sont définies comme suit :

— **Rouge** : GPIO 17 (Pin 11)
— **Orange** : GPIO 27 (Pin 13)
— **Vert** : GPIO 22 (Pin 15)

Chaque LED est associée à une résistance en série afin de limiter le courant et éviter d'endommager les composants. La communication avec un second Raspberry Pi, installé dans le véhicule, est assurée par un point d'accès Wi-Fi. Ce dernier reçoit les mises à jour de l'état du feu en utilisant le protocole MQTT et les affiche sur une interface utilisateur dédiée. L'ensemble du système est alimenté par la machine hôte, garantissant un fonctionnement stable et sécurisé.

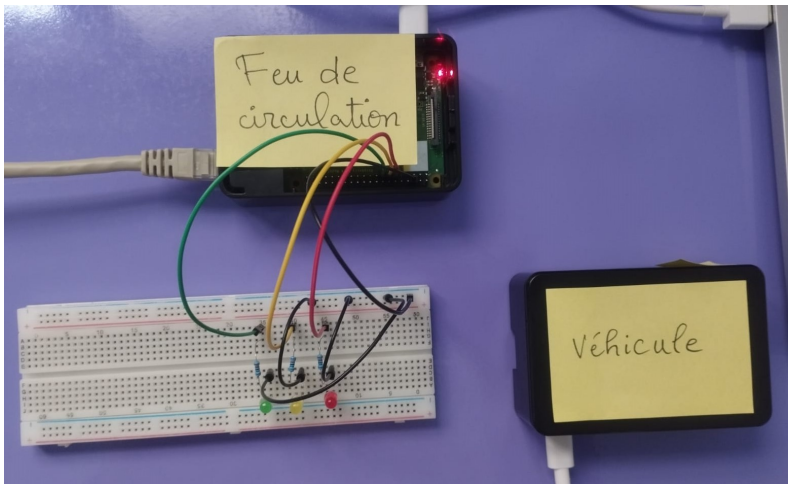

FIGURE 2.3.2 – Le circuit du projet.

2.4 Configuration des appareils

Dans le cadre de notre projet, nous avons configuré trois appareils principaux : un Raspberry Pi installé sur le feu tricolore, un Raspberry Pi embarqué dans le véhicule, et un PC utilisé comme broker MQTT pour assurer la communication entre les deux premiers. Cette configuration nous a permis d'établir un système de communication sans fil efficace et en temps réel.

2.4.1 Raspberry Pi 1 (Feux de Circulation)

Nous avons commencé par configurer le premier Raspberry Pi, qui joue un rôle central dans la gestion du feu tricolore. Cette unité est chargée de surveiller en permanence l'état du feu et de transmettre ces informations via MQTT.

Tout d'abord, nous avons installé Raspberry Pi OS sur une carte microSD, puis nous avons activé le SSH et la connexion Wi-Fi afin de pouvoir contrôler le Raspberry Pi à distance. Après cela, nous avons mis à jour le système et installé les bibliothèques nécessaires pour la gestion des GPIOs et la communication MQTT.

Nous avons ensuite procédé au câblage du feu tricolore en connectant trois LEDs (rouge, orange et verte) aux broches GPIO du Raspberry Pi. Chaque LED a été associée à une résistance pour limiter le courant et éviter d'endommager les composants.
Après avoir vérifié les connexions, nous avons installé Mosquitto et Paho-MQTT, deux outils essentiels pour l'implémentation du protocole MQTT, permettant la communication entre les différents appareils.

```
sudo apt update && sudo apt upgrade -y
sudo apt install python3-pip
pip3 install paho-mqtt RPi.GPIO
```

2.4.2 Raspberry Pi 2 (Véhicule)

Le second Raspberry Pi, installé dans le véhicule, est responsable de la réception des informations envoyées par le premier et de leur affichage sur une interface utilisateur.
Nous avons suivi les mêmes étapes d'installation et de configuration que pour le premier Raspberry Pi :

1. Installation de Raspberry Pi OS
2. Activation du SSH et du Wi-Fi
3. Mise à jour du système
4. Installation de Mosquitto et Paho-MQTT

En plus de ces éléments, nous avons créer un environnement python virtuel surlequel on a installé Flask et Flask-SocketIO, qui permettent de développer une interface web interactive. Cette interface est accessible depuis un navigateur et affiche l'état du feu en temps réel.

```
sudo apt install python3-venv
python3 -m venv venv
source venv/bin/activate
pip install flask flask-socketio paho-mqtt
```

2.4.3 Machine (Broker)

Le PC joue le rôle de broker MQTT, c'est-à-dire qu'il gère l'échange des messages entre les deux Raspberry Pi. Nous avons installé Mosquitto, qui est un serveur MQTT open-source, afin d'assurer cette fonction. L'installation a été réalisée via les commandes suivantes :

```
sudo apt update
sudo apt install mosquitto mosquitto-clients
```

Nous avons ensuite activé et démarré le service Mosquitto :

```
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

2.4.4 Test de la communication

Pour tester la configuration, nous avons utilisé deux terminaux sur le PC :

1. Dans le premier terminal, nous avons lancé un client Mosquitto en mode abonnement sur le Raspberry Pi embarqué dans le véhicule (Rasp2) pour écouter les messages publiés sur le sujet traffic/light :

```
mosquitto_sub -h localhost -t "traffic/light"
```

2. Dans le deuxième terminal, nous avons publié un message de test depuis le Raspberry Pi du feu tricolore (Rasp1) sur le même sujet :

```
mosquitto_pub -h localhost -t "traffic/light" -m "Red"
```

La communication a été correctement établie : le message "Red" est apparu instantanément sur le terminal du Raspberry Pi embarqué dans le véhicule, confirmant ainsi la bonne transmission des données via MQTT.

2.5 Codage

2.5.1 Raspberry Pi (Feu tricolore - Émetteur MQTT)

Une fois la configuration matérielle et logicielle terminée, nous avons commencé par développer le code permettant au Raspberry Pi du feu tricolore d'envoyer en temps réel l'état du feu. Pour cela, nous avons utilisé Python avec les bibliothèques RPi.GPIO (pour contrôler les LEDs) et paho-mqtt (pour envoyer les messages MQTT).
Le script fonctionne en boucle et change l'état des LEDs toutes les dix secondes en envoyant un message correspondant ("Red", "Orange" ou "Green") au broker MQTT.

```
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import time

# Configuration des broches GPIO
RED_LED = 17
ORANGE_LED = 27
GREEN_LED = 22

GPIO.setmode(GPIO.BCM)
GPIO.setup(RED_LED, GPIO.OUT)
GPIO.setup(ORANGE_LED, GPIO.OUT)
GPIO.setup(GREEN_LED, GPIO.OUT)

# Configuration MQTT
BROKER = "192.168.55.52"  # Adresse IP du broker (PC)
TOPIC = "traffic_light/state"
client = mqtt.Client()

# Fonction pour changer les feux
```

```
def change_light(state):
    GPIO.output(RED_LED, GPIO.HIGH if state == "red" else GPIO.LOW)
    GPIO.output(ORANGE_LED, GPIO.HIGH if state == "orange" else GPIO.LOW)
    GPIO.output(GREEN_LED, GPIO.HIGH if state == "green" else GPIO.LOW)

# Connexion au broker
client.connect(BROKER, 1883, 60)

# Durée de chaque signal
durations = {
    "red": 10,
    "orange": 10,
    "green": 10
}

# Simulation des feux de circulation
try:
    while True:
        for state in ["red", "orange", "green"]:
            change_light(state)
            message = f'{{"state": "{state}"}}'
            client.publish(TOPIC, message)
            print("Message envoyé :", message)
            time.sleep(5)
except KeyboardInterrupt:
    GPIO.cleanup()
```

2.5.2 Raspberry Pi (Véhicule - Abonné MQTT)

Du côté du véhicule, nous avons développé un programme qui s'abonne au topic "traffic/light" du broker MQTT et qui met à jour une interface web en fonction des messages reçus.
Nous avons mis en place un serveur Flask qui récupère les informations en temps réel et les transmet via SocketIO à une page web. Cette dernière affiche dynamiquement l'état du feu, permettant ainsi au conducteur de visualiser les changements instantanément.

```
import paho.mqtt.client as mqtt
from flask import Flask, render_template
from flask_socketio import SocketIO
import json

app = Flask(__name__)
socketio = SocketIO(app)

# Valeur initiale de l'état du feu
traffic_state = "unknown"

# Callback pour recevoir les messages MQTT
def on_message(client, userdata, msg):
```

```
    global traffic_state
    try:
        message = json.loads(msg.payload.decode())
        state = message.get("state", "unknown")
        if state != "unknown":
            traffic_state = state
        print(f"Message reçu : {state}")
        # Emettre l'état du feu en temps réel sur le client via SocketIO
        socketio.emit('update_state', {'state': state})
    except json.JSONDecodeError:
        print("Erreur : Impossible de décoder le message reçu.")

# Configuration MQTT
BROKER = "192.168.55.52"  # Adresse IP du broker (PC)
TOPIC = "traffic_light/state"
client = mqtt.Client()
client.on_message = on_message
client.connect(BROKER, 1883, 60)
client.subscribe(TOPIC)
client.loop_start()

# Route Flask pour afficher l'état des feux
@app.route("/")
def home():
    global traffic_state
    if traffic_state == "unknown":
        return render_template("index.html", state="En attente des données")
    return render_template("index.html", state=traffic_state)

# Lancer l'application Flask avec SocketIO
if __name__ == "__main__":
    socketio.run(app, host="0.0.0.0", port=5000)
```

2.6 Validation

2.6.1 Test de la Gestion des Feux Tricolores

Nous avons exécuté notre script de contrôle des feux sur le Raspberry Pi du feu tricolore :

```
sudo python3 traffic_light_publisher.py
```

Nous avons observé le comportement des LEDs pour vérifier qu'elles s'allumaient et s'éteignaient chacune pour la durée consacrée dans un cycle répétitif.
Nous avons confirmé en même temps que les messages envoyés correspondaient bien aux changements des feux en temps réel, avec des valeurs comme "Red", "Orange", et "Green" qui s'affichaient successivement sur le terminal.

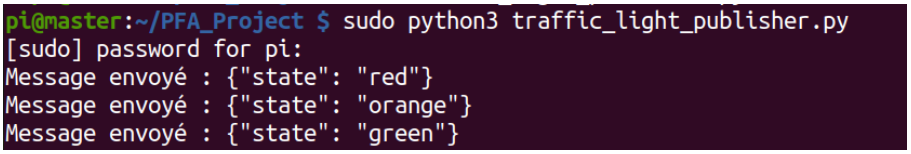

FIGURE 2.6.3 – Terminal du Raspberry Pi (Feu tricolore).

2.6.2 Test de la Réaction du Raspberry Pi du Véhicule aux Messages MQTT

Pendant que le script de contrôle des feux tournait, nous avons rouvert un terminal sur le Raspberry Pi du véhicule et exécuté notre script d'abonnement sur le Raspberry Pi du véhicule :

```
python3 traffic_light_subscriber.py
```

Nous avons vérifié dans le terminal que chaque message reçu entraînait l'affichage d'un changement d'état dans le programme.

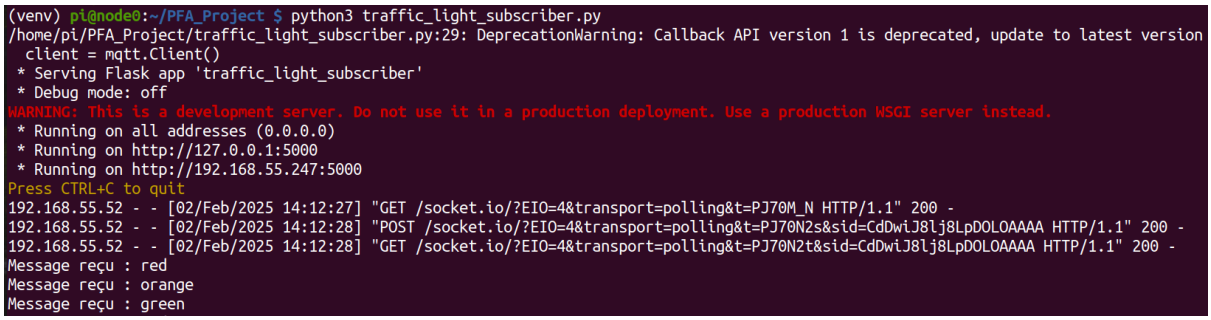

FIGURE 2.6.4 – Terminal du Raspberry Pi (Véhicule).

2.7 Conclusion

À travers les différentes étapes de configuration, de codage et de test, nous avons mis en place un système fonctionnel assurant la communication entre un feu tricolore et un véhicule à l'aide du protocole MQTT. La configuration des Raspberry Pi, du broker MQTT, et du réseau a permis d'établir une communication fiable. Le développement du code a ensuite permis de gérer l'envoi et la réception des états du feu en temps réel. Enfin, les tests ont validé le bon fonctionnement du système, en confirmant que les messages sont correctement transmis, reçus et interprétés. Ces résultats démontrent la fiabilité et la réactivité du système avant l'intégration de l'interface utilisateur.

Chapitre 3

Interface homme machine du projet

3.1 Introduction

Ce chapitre a pour objectif de présenter l'Interface Homme-Machine (IHM) développée pour afficher en temps réel l'état des feux de circulation au conducteur. Conçue pour améliorer la sécurité et la fluidité du trafic, elle offre une interface intuitive et réactive. Nous aborderons ses spécifications, son architecture, les technologies utilisées ainsi que son intégration et son évaluation.

3.2 L'IHM

3.2.1 Introduction à l'IHM

L'IHM facilite l'interaction entre l'utilisateur et un système embarqué, jouant un rôle clé dans les véhicules connectés pour améliorer l'expérience utilisateur, la sécurité et la fluidité du trafic. Une conception optimisée réduit la charge cognitive et accélère la prise de décision.
Dans ce projet, l'IHM notifie en temps réel l'état du feu via une application intuitive, minimisant les arrêts brusques et optimisant la consommation de carburant. Grâce à l'IoT et au protocole MQTT, elle assure une communication fiable entre l'infrastructure et le véhicule, contribuant aux smart cities et aux transports intelligents.

3.2.2 Spécifications Fonctionnelles de l'IHM

Besoins fonctionnels

Les besoins fonctionnels sont les besoins qui déterminent les fonctionnalités indispensables auxquels doit répondre notre solution. L'IHM en sa globalité doit répondre aux exigences fonctionnelles suivantes :

— *Affichage en temps réel de l'état du feu de circulation :* L'application doit informer le conducteur de l'état actuel du feu (rouge, orange ou vert) dès que le véhicule approche d'un carrefour.
— *Envoi de notifications instantanées :* Le conducteur doit recevoir une alerte visuelle et/ou sonore à l'approche d'un feu tricolore pour anticiper ses actions.
— *Messages de guidage dynamiques :* L'application doit fournir des recommandations basées sur l'état du feu, pour améliorer la prise de décision du conducteur.

Besoins non fonctionnels

Les besoins non fonctionnels définissent les critères de qualité que l'Interface Homme-Machine (IHM) doit respecter pour assurer une utilisation efficace et sécurisée. Ils garantissent la fiabilité, la robustesse et la réactivité du système, permettant une interaction fluide entre le conducteur et l'infrastructure routière. Parmi ces exigences, nous avons :

— *Fiabilité et disponibilité :* L'application doit fonctionner en continu et fournir des informations précises sans interruption pour éviter toute confusion ou retard dans la prise de décision du conducteur.
— *Réactivité et faible latence :* La communication entre le véhicule et le feu de circulation via MQTT doit être rapide, avec un temps de réponse minimal pour garantir l'affichage instantané de l'état du feu.
— *Ergonomie et facilité d'utilisation :* L'interface doit être simple, intuitive et bien conçue pour permettre au conducteur de comprendre rapidement les informations sans être distrait.
— *Robustesse et résilience :* L'application doit pouvoir fonctionner correctement malgré d'éventuelles perturbations du réseau Wifi ou des conditions météorologiques défavorables.

3.3 Conception de l'IHM

3.3.1 Architecture de l'IHM

L'architecture de l'IHM combine matériel et logiciel pour une interaction fluide entre le conducteur et le système de signalisation. Un Raspberry Pi placé au niveau du feu de circulation et du véhicule assure la communication via MQTT sur WiFi. Un écran embarqué (tablette, tableau de bord ou smartphone) affiche l'état du feu en temps réel.
L'IHM est une interface web dynamique développée avec HTML, CSS et JavaScript, intégrant MQTT.js pour recevoir et traiter les données du feu de circulation. Elle affiche graphiquement les feux et fournit des messages de guidage au conducteur pour anticiper les changements de signalisation.
Cette architecture garantit une mise à jour instantanée des signaux, améliorant la réactivité et la fluidité du trafic. Des schémas peuvent illustrer le système, incluant la connexion Raspberry Pi - MQTT - Application web, l'interaction logicielle, et un flux de communication MQTT.

3.3.2 Diagramme de cas d'utilisation

Le diagramme des cas d'utilisation est un modèle simplifié du fonctionnement de l'application qui spécifie les acteurs qui l'utilisent et les services qu'il leur offre.
Le diagramme de la figure ci-dessous illustre les rôles des acteurs ainsi que l'étendue de leur responsabilités.

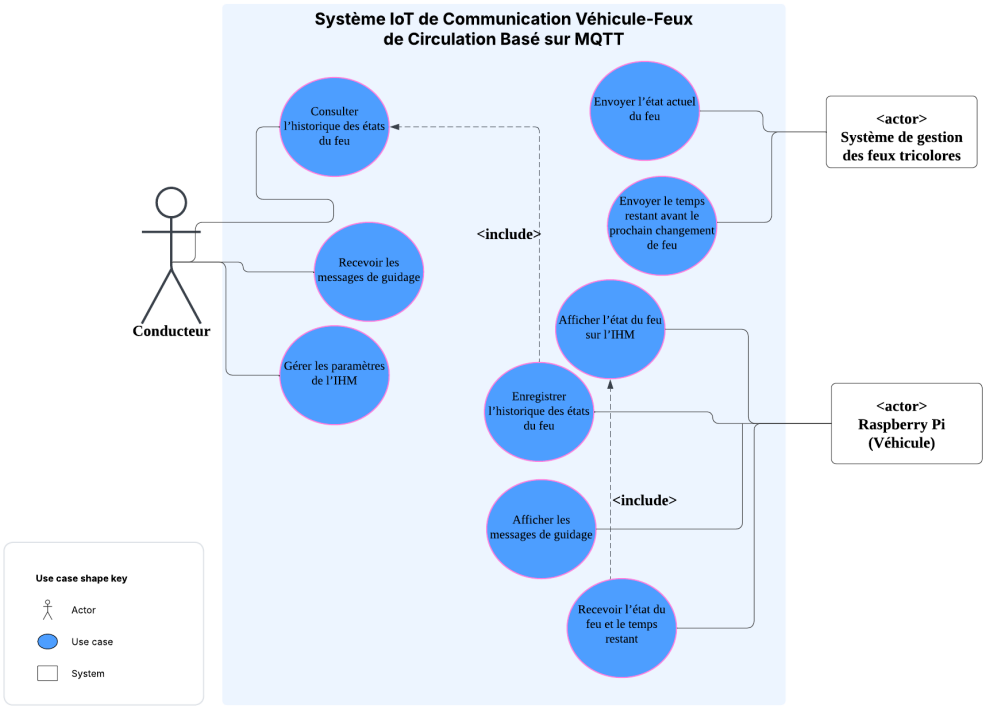

FIGURE 3.3.1 – Diagramme de cas d'utilisation.

3.3.3 Technologies utilisées

Dans cette partie, nous nous intéressons au langage, aux bibliothèques et aux techniques de programmation utilisées dans la réalisation de notre projet.

— **HTML :** est le langage de balisage standard utilisé pour créer des pages web. Il décrit la structure d'une page web en utilisant des éléments HTML, qui indiquent au navigateur comment afficher le contenu. Ces éléments définissent des parties spécifiques du contenu comme les titres, les paragraphes, et les liens.[6]

FIGURE 3.3.2 – Logo d'HTML.[7]

— **CSS :** est un langage permettant de styliser les pages web en définissant l'apparence des éléments HTML (comme les couleurs, les polices, et la mise en page). Il sépare le contenu de la présentation, facilitant ainsi la gestion du design d'un site. Grâce à des fichiers CSS externes, on peut appliquer un style cohérent à plusieurs pages d'un site web.[6]

FIGURE 3.3.3 – Logo de CSS.[7]

— **JavaScript :** est un langage de programmation largement utilisé pour développer des comportements dynamiques sur les pages web. Il permet d'ajouter des interactions, de manipuler le contenu et d'effectuer des actions en réponse aux événements utilisateurs. JavaScript est essentiel pour le développement web, aux côtés de HTML et CSS, en permettant de rendre les sites plus interactifs et fonctionnels.[6]

FIGURE 3.3.4 – Logo de JavaScript. [7]

— **Flask Socket.IO :** Flask est un micro-framework web en Python, apprécié pour sa simplicité et sa flexibilité. Socket.IO, une bibliothèque JavaScript, permet une communication bidirectionnelle en temps réel entre clients et serveur. L'intégration de Flask avec Socket.IO facilite le développement d'applications interactives mises à jour instantanément, sans rechargement de page, comme les chats en ligne, les jeux ou les outils collaboratifs.[8]

FIGURE 3.3.5 – Logo de Flask Socket.IO. [9]

3.4 Communication et Intégration avec le Système

L'intégration du système repose sur la communication en temps réel entre le Raspberry Pi et l'interface utilisateur via le protocole MQTT. Cette architecture assure une transmission efficace des mises à jour du feu tricolore et une réactivité optimale pour le conducteur.

3.4.1 Connexion avec le Raspberry Pi

Le Raspberry Pi, connecté aux feux de circulation, publie en temps réel l'état du feu (vert, orange ou rouge) sur un broker MQTT. L'interface web, développée avec Flask et Flask-SocketIO, s'abonne à ces messages pour afficher dynamiquement les informations reçues. Grâce à cette communication bidirectionnelle, l'interface réagit instantanément

aux changements de signalisation.
L'interface web développée avec Flask et Flask-SocketIO s'abonne au topic MQTT pour recevoir les mises à jour du feu en temps réel.
L'interface Flask s'abonne au topic traffic light/status et met à jour l'affichage en fonction de l'état du feu.

3.4.2 Interaction utilisateur

L'interface embarquée sur le véhicule affiche clairement l'état actuel du feu et informe le conducteur via des notifications visuelles. Cette interaction intuitive permet d'anticiper les changements de signalisation, réduisant ainsi les arrêts brusques et optimisant la fluidité du trafic. En assurant une mise à jour automatique et sans rechargement de la page, l'IHM garantit une expérience utilisateur fluide et efficace.

3.5 Développement et implémentation

Dans cette section, nous détaillons le développement et l'implémentation de notre Interface Homme-Machine (IHM), en présentant ses fonctionnalités principales et son fonctionnement pour assurer une communication fluide et en temps réel entre les feux de circulation et les véhicules.

FIGURE 3.5.6 – Interface Homme Machine.

L'Interface Homme-Machine (IHM) affiche en temps réel l'état des feux de circulation (vert, orange, rouge) pour informer les conducteurs. Elle indique clairement quand continuer (feu vert), ralentir (feu orange) ou s'arrêter (feu rouge), offrant une interface intuitive

et facile à lire. Cette conception vise à améliorer la réactivité et la sécurité des conducteurs, favorisant une conduite fluide et anticipée.

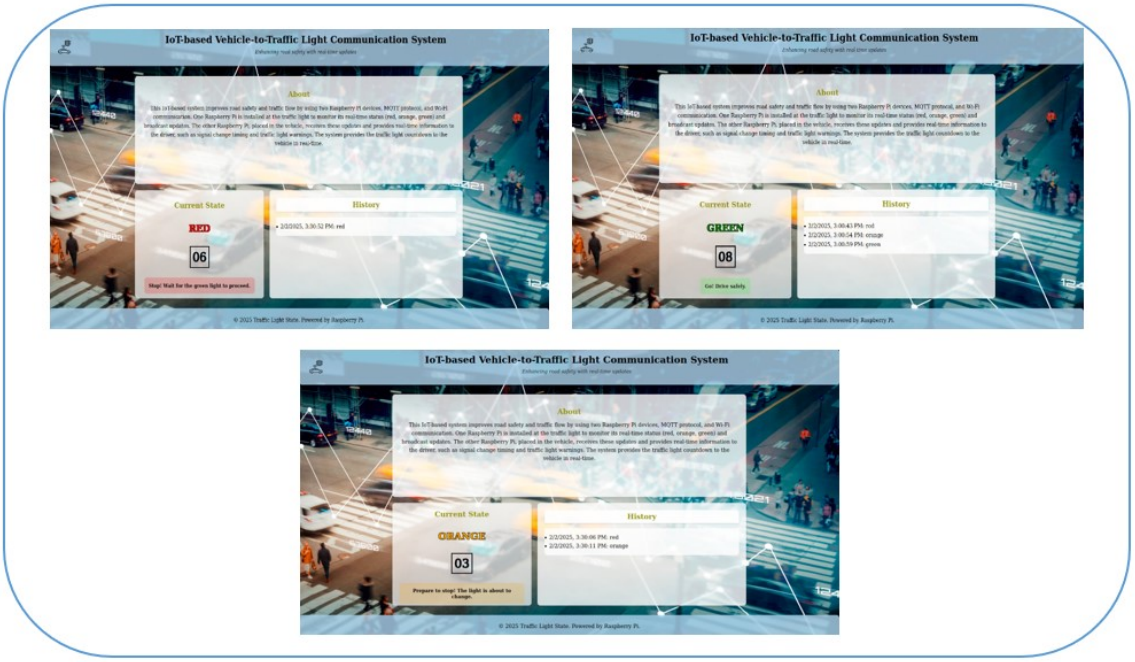

FIGURE 3.5.7 – Représentation de l'IHM pour les différents états des feux de circulation.

3.6 Conclusion

Tout au long de ce chapitre, nous avons mis en avant l'importance de l'Interface Homme-Machine (IHM) dans notre projet. En assurant une communication claire et intuitive entre le véhicule et les feux de circulation, elle contribue à une conduite plus fluide et sécurisée. Son intégration avec les technologies IoT et le protocole MQTT démontre son efficacité dans un système de transport intelligent.

Conclusion générale

Ce projet a permis la mise en place d'un système V2I (Vehicle-to-Infrastructure) basé sur l'IoT et utilisant le protocole MQTT pour améliorer la sécurité routière et optimiser la gestion du trafic. Grâce à l'intégration de deux Raspberry Pi, l'un installé au niveau d'un feu tricolore et l'autre embarqué dans un véhicule, le système assure une communication en temps réel permettant aux conducteurs de recevoir des mises à jour instantanées sur l'état du feu. L'interface web développée avec Flask et Flask-SocketIO facilite l'affichage des informations, offrant ainsi une meilleure expérience utilisateur.

L'architecture mise en place a démontré son efficacité en garantissant un échange fluide et fiable des données via un réseau Wi-Fi, tout en respectant les exigences de réactivité essentielles aux applications de transport intelligent. L'implémentation et les tests réalisés ont confirmé la bonne transmission des messages MQTT et le fonctionnement optimal de l'interface utilisateur.

Ce projet ouvre la voie à plusieurs améliorations et évolutions possibles. Une extension naturelle serait l'intégration de la technologie V2V (Vehicle-to-Vehicle) pour permettre la communication directe entre véhicules, renforçant ainsi la sécurité et la fluidité du trafic. Par exemple, les véhicules pourraient échanger des informations sur leur vitesse, leur position ou des conditions de circulation particulières, permettant ainsi une prise de décision plus intelligente et collaborative.

En somme, cette solution V2I constitue une première étape vers un système de transport intelligent (ITS) plus avancé, où l'échange d'informations entre infrastructures et véhicules, et potentiellement entre véhicules eux-mêmes, contribuera à améliorer la sécurité et l'efficacité des déplacements urbains.

Bibliographie

[1] https ://igm.univ-mlv.fr/ dr/XPOSE2008/SCRUM/presentation.php (consulté le 1/2/2025)

[2] http ://www.agileforall.com (consulté le 1/2/2025)

[3] https ://www.journaldunet.fr/web-tech/dictionnaire-de-l-iot/1440686-mqtt-comment-fonctionne-ce-protocole/ (consulté le 1/2/2025)

[4] https ://www.paessler.com/fr/it-explained/mqtt (consulté le 1/2/2025)

[5] https ://aws.amazon.com/fr/what-is/mqtt/ (consulté le 1/2/2025)

[6] https ://www.w3schools.com (consulté le 2/2/2025)

[7] https ://www.flaticon.com (consulté le 2/2/2025)

[8] https ://www.videosdk.live/developer-hub/socketio/flask-socketio (consulté le 2/2/2025)

[9] https ://pythonprogramminglanguage.com/python-flask-websocket/ (consulté le 2/2/2025)