Simulation de Déplacement de Véhicules avec Communication via MQTT

Auteur : Ahmadou Bassirou DIALLO

Date : 27/05/2024

Affiliation : Université de Reims Champagne-Ardenne

TABLE DES MATIERES

1. INTRODUCTION

Ce projet a pour objectif de simuler le déplacement de véhicules en utilisant des files MQTT pour la communication. Chaque véhicule publie sa position sur une file MQTT et peut recevoir des requêtes pour sa position actuelle et son horloge. Le projet comprend également un upper-tester pour tester le bon fonctionnement des boitiers de communication des véhicules.

2. ALGORITHMES

A. Déplacement du Véhicule

Le script **vehicle_script.py** simule le déplacement d'un véhicule en suivant une série d'étapes prédéfinies. À chaque étape, le véhicule met à jour sa position et publie cette nouvelle position sur la file MQTT vehicle.

a. **Initialisation** : Lecture des paramètres initiaux du véhicule (position, direction, vitesse) à partir d'un fichier JSON.
b. **Lecture des Étapes** : Chargement des étapes de déplacement à partir d'un fichier JSON.
c. **Déplacement** : Calcul de la nouvelle position en fonction de la direction et de la vitesse, puis publication de cette position sur la file MQTT vehicle.
d. **Réception des Requêtes** : Réception des requêtes de position et d'horloge via la file MQTT request et réponse avec la position actuelle et l'horloge.

Description Code

```
def on_message(client, userdata, message):
    global next_destination
    data = json.loads(message.payload.decode())
    if message.topic == "vehicle":
        print(data['position'])
        if data['position'] == next_destination:
            print(f"Prochaine destination ({next_destination}) est occupée par
le véhicule {data['id']}. En attente...")
        else:
            print(f"Aucun véhicule occupant la prochaine destination
({next_destination}). Déplacement autorisé.")
    elif message.topic == "request":
        if data['request'] == 'get_status' and data['id'] == vehicle['id']:
            response = {
                'id': vehicle['id'],
                'position': vehicle['position'],
                'timestamp': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            }
            client.publish("response", json.dumps(response))
client.on_message = on_message
```

Cette fonction `**on_message**` est une fonction callback qui est déclenchée à chaque fois qu'un message MQTT est reçu. Elle vérifie le sujet du message pour déterminer s'il provient d'un véhicule ou s'il s'agit d'une requête. Si le message concerne un véhicule, elle vérifie si la position du véhicule correspond à la prochaine destination. Si c'est le cas, elle imprime un message indiquant que la destination est occupée. Sinon, elle autorise le déplacement. Si le message est une requête de statut, elle répond en envoyant l'ID du véhicule, sa position actuelle et l'horodatage.

```
def read_init_params(file_path):
    with open(file_path) as f:
        init_params = json.load(f)
    return init_params['id'], init_params['x'], init_params['y'],
init_params['dir'], init_params['speed']
```

Cette fonction `**read_init_params**` lit les paramètres initiaux d'un véhicule à partir d'un fichier JSON et retourne ces paramètres sous forme de tuple, comprenant l'ID du véhicule, sa position (x, y), sa direction et sa vitesse.

```
def read_steps(file_path):
    with open(file_path) as f:
        steps = json.load(f)
    return steps['steps']
```

Cette fonction lit les étapes de déplacement à partir d'un fichier JSON spécifié. Elle ouvre le fichier, charge les données JSON et extrait la liste des étapes de déplacement. Enfin, elle retourne cette liste d'étapes.

```
def move_vehicle(vehicle, speed, direction):
    x, y = vehicle['position']
    if direction == 1:
        x += speed
    elif direction == -1:
        x -= speed
    elif direction == 2:
        y += speed
    elif direction == -2:
        y -= speed
    vehicle['position'] = (x, y)
    client.publish("vehicle", json.dumps({'id': vehicle['id'], 'position': (x,
y)}))
```

Cette fonction prend en paramètres un véhicule, une vitesse et une direction, puis met à jour la position du véhicule en fonction de ces informations. Elle calcule la nouvelle position en ajoutant ou en soustrayant la vitesse à la position actuelle du véhicule selon la direction spécifiée. Ensuite, elle met à jour la position du véhicule dans le dictionnaire du véhicule et publie la nouvelle position sur le topic "vehicle" via MQTT en format JSON, avec l'identifiant du véhicule et ses nouvelles coordonnées.

```
for step in steps:
    # Extraire la vitesse et la direction de l'étape
    speed, direction = step['speed'], step['direction']

    # Avancer le véhicule avec les paramètres de cette étape
    move_vehicle(vehicle, speed, direction)

    # Afficher la position du véhicule
    print(f"Véhicule {vehicle['id']} - Position : {vehicle['position']}")
```

```
    # Déterminer la prochaine destination
    next_destination = (vehicle['position'][0] + (speed if direction in [1,
-1] else 0),
                        vehicle['position'][1] + (speed if direction in [2,
-2] else 0))

    # Attendre un certain temps avant de vérifier à nouveau la file vehicle
    time.sleep(5)
```

Ce bloc de code itère sur chaque étape de déplacement spécifiée dans la liste `steps`. Pour chaque étape, il extrait la vitesse et la direction, puis appelle la fonction `**move_vehicle**` pour avancer le véhicule avec ces paramètres. Ensuite, il affiche la position mise à jour du véhicule, détermine la prochaine destination en fonction de la vitesse et de la direction, puis attend un certain temps avant de passer à l'étape suivante. Cette pause est nécessaire pour simuler le déplacement du véhicule à intervalles réguliers.

B. Test du Véhicule (Upper-Tester)

Le script **upper_tester_script.py** envoie des requêtes à un véhicule spécifique pour obtenir sa position actuelle et l'horloge.

a. **Envoi de Requête :** Envoi d'une requête sur la file MQTT request avec l'ID du véhicule cible.
b. **Réception de Réponse :** Écoute des réponses sur la file MQTT response et affichage des données reçues.

Description Code

```
def on_message(client, userdata, message):
    data = json.loads(message.payload.decode())
    if message.topic == "response":
        print(f"Réponse reçue : {data}")
        responses.append(data)

client.on_message = on_message
def send_request(vehicle_id):
    request = {
        'request': 'get_status',
        'id': vehicle_id
    }
    client.publish("request", json.dumps(request))
```

La fonction **on_message** est appelée lorsque des messages sont reçus. Si le message provient du topic "**response**", il affiche et stocke la réponse reçue dans la liste responses.

La fonction **send_request** envoie une requête au véhicule spécifié par **vehicle_id**, demandant sa position et la valeur de son horloge.

C. Utilisation des Files MQTT

Les files MQTT permettent la communication asynchrone entre les véhicules et l'upper-tester. Les principaux topics utilisés sont :

- **vehicle** : Pour publier les positions des véhicules.
- **request** : Pour envoyer des requêtes de position et d'horloge aux véhicules.
- **response** : Pour recevoir les réponses des véhicules aux requêtes.

Échanges

- **Véhicule à Upper-Tester** : Publication de la position actuelle sur vehicle.
- **Upper-Tester à Véhicule** : Envoi d'une requête de position et d'horloge sur request.
- **Véhicule à Upper-Tester** : Réponse avec la position actuelle et l'horloge sur response.

D. Copies d'Écran d'Exécution

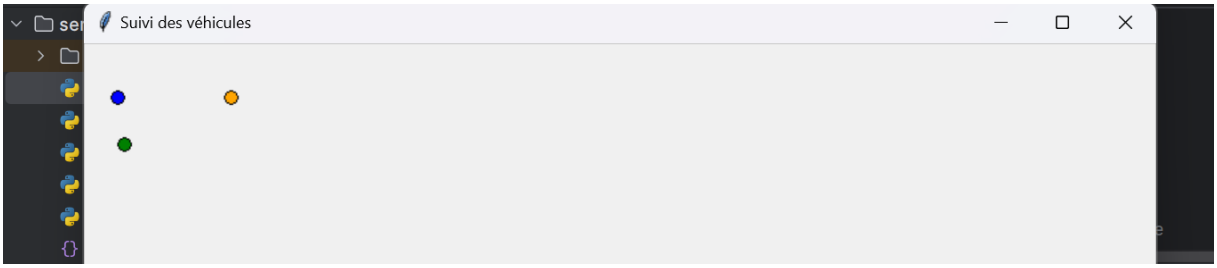

Figure 1 : Suivi et Contrôle des Véhicules à l'aide de MQTT et d'une Interface Graphique Tkinter

```
Aucun véhicule occupant la prochaine destination ((210, 50)). Déplacement autorisé.
Véhicule 3 - Position : (110, 70)
[110, 70]
Aucun véhicule occupant la prochaine destination ((110, 90)). Déplacement autorisé.
Véhicule 3 - Position : (110, 75)
[110, 75]
Aucun véhicule occupant la prochaine destination ((110, 80)). Déplacement autorisé.
Véhicule 3 - Position : (110, 35)
[110, 35]
Aucun véhicule occupant la prochaine destination ((110, 75)). Déplacement autorisé.
Véhicule 3 - Position : (110, 40)
[110, 40]
Aucun véhicule occupant la prochaine destination ((110, 45)). Déplacement autorisé.

(venv) C:\Users\mouba\OneDrive\Bureau\Mobiles_app\car\server>
```

Figure 2 : Déplacement du véhicule 3

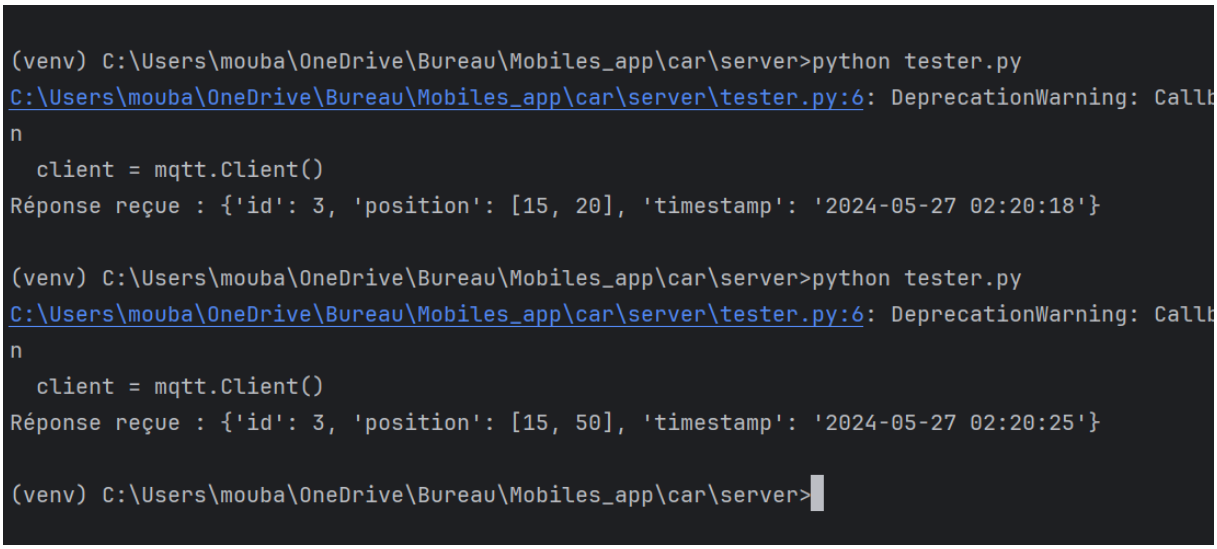

Figure 3 : Capture d'écran du Résultat de l'Exécution de l'Upper-Tester

3. LIEN GIT

Le code source complet de ce projet est disponible pour consultation et contribution sur notre dépôt GitHub. Accédez au projet en suivant ce lien : Accéder au Projet Git

4. CONCLUSION

Ce projet a démontré avec succès l'intégration et l'interaction entre des véhicules intelligents via un système de communication basé sur MQTT. En exploitant les capacités de MQTT pour la transmission de messages en temps réel, nous avons développé une application capable de gérer la position et l'état des véhicules, tout en respectant les signaux de feux de circulation simulés.