

Université Euro-Méditerranéenne de Fès
Ecole d'Ingénierie Digitale et d'Intelligence Artificielle

Rapport

Module : Real Time Operating Systems (RTOS)

Contrôle Temps Réel d'un Robot Mobile :
ESP32 (Wi-Fi) ↔ STM32 (FreeRTOS)

Encadré par : *Pr Yahia MAZZI*

Réalisé par :

Bouarfa LAHMAR

Hiba MOUHSINE

Mohamed BOUFARHA

Année Universitaire :

2024 // 2025

Table des matières

1.	Introduction :	3
1.1.	Contexte du Projet	3
1.2.	Objectifs pédagogiques et techniques	3
1.3.	Technologies utilisées	4
2.	Architecture générale du système	5
2.1.	Présentation globale	5
2.2.	Schéma d'architecture fonctionnelle	5
3.	Fonctionnalités du système embarqué	7
3.1.	Fonctionnalités côté ESP32	7
3.1.1.	Connexion Wi-Fi	7
3.1.2.	Serveur HTTP local	7
3.1.3.	Communication UART avec le STM32	7
3.1.4.	Lecture du capteur de vitesse (FC-03)	8
3.1.5.	Calcul de la position (X, Y)	8
3.1.6.	Envoi des données vers le serveur Flask	8
3.2.	Fonctionnalités côté STM32	9
3.2.1.	Réception UART des commandes	9
3.2.2.	Contrôle des moteurs via L298N	9
3.2.3.	Structure des tâches FreeRTOS	9
3.2.4.	Gestion des interruptions	10
3.2.5.	Envoi des données vers l'ESP32	10
4.	Serveur Flask et Interface Web	10
4.1.	Serveur Flask	10
4.2.	Interface Web	11
5.	Communication entre les composants	11
5.1.	Communication ESP32 ↔ STM32 (UART)	11
5.2.	Communication ESP32 ↔ Serveur Flask (HTTP)	11
5.3.	Communication Serveur Flask ↔ Interface Web (WebSocket)	12
6.	Conclusion	12

1. Introduction :

1.1. Contexte du Projet

Avec l'évolution rapide des technologies embarquées et de la connectivité sans fil, les systèmes cyber-physiques deviennent de plus en plus répandus dans les domaines de l'automatisation, de la robotique et de l'Internet des Objets (IoT). Ces systèmes combinent des capacités de traitement embarqué avec des mécanismes de communication réseau, permettant le contrôle à distance, la supervision en temps réel, et l'échange bidirectionnel d'informations.

Dans ce contexte, les microcontrôleurs tels que l'ESP32 et le STM32 offrent une plateforme puissante pour le développement de systèmes embarqués distribués. Le premier, avec ses capacités Wi-Fi et Bluetooth intégrées, permet une communication réseau fluide, tandis que le second, optimisé pour le traitement temps réel, est parfaitement adapté à la gestion des actionneurs, des capteurs, et de la logique de contrôle.

Le projet ici présenté consiste à concevoir un robot embarqué capable de recevoir des commandes à distance, de les transmettre à un sous-système dédié au pilotage moteur, puis de renvoyer régulièrement des données relatives à sa vitesse et à sa position à un serveur web. Ce système est divisé en deux composantes principales : un ESP32 pour la communication, la gestion des capteurs et la transmission des données ; et un STM32 pour la gestion temps réel des moteurs via le système d'exploitation temps réel FreeRTOS.

Ce projet illustre ainsi une intégration complète des aspects essentiels des systèmes embarqués modernes : gestion réseau, traitement multitâche temps réel, communication inter-processeur, interface web, et traitement de capteurs.

1.2. Objectifs pédagogiques et techniques

Ce projet a pour ambition de renforcer les compétences en développement de systèmes embarqués temps réel, en intégrant à la fois des volets matériels et logiciels.

Les objectifs techniques du projet sont les suivants :



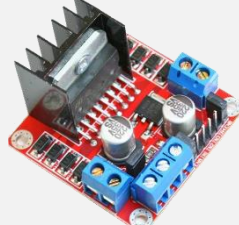

- Implémenter une connexion réseau Wi-Fi stable sur l'ESP32 ;
- Déployer un serveur HTTP local capable de recevoir des commandes distantes ;
- Mettre en place une communication série (UART) bidirectionnelle entre ESP32 et STM32 ;
- Lire les données d'un capteur de vitesse MP 6050 pour mesurer la distance parcourue ;
- Calculer la position (X, Y) et l'orientation du robot en temps réel ;
- Concevoir un serveur web Flask pour piloter le robot et afficher ses données ;
- Développer un système multitâche sous FreeRTOS sur STM32 pour exécuter les commandes moteurs et assurer la réactivité du robot.

D'un point de vue pédagogique, le projet permet d'apprendre à :

- Concevoir une architecture embarquée modulaire et communicante ;
- Gérer les contraintes de temps réel dans un environnement multi-tâches ;
- Intégrer des protocoles de communication (UART, HTTP, WebSocket) ;
- Visualiser et interpréter des données robotiques via une interface web interactive.

1.3. Technologies utilisées

Le projet mobilise plusieurs briques matérielles et logicielles que nous avons dû configurer, interfacer et tester ensemble :

Catégorie	Composant	Image illustrative
Microcontrôleur	ESP32 DevKit jouant le rôle de passerelle Wi-Fi et UART	
Microcontrôleur	STM32F429I responsable du contrôle moteur temps réel.	
Contrôle moteur	L298N Driver permet de piloter les moteurs en contrôlant leur direction et leur vitesse via les broches de contrôle du STM32.	
Capteur	FC-03 (capteur de vitesse) permettant de calculer la distance parcourue	

Catégorie	Logiciel
IDE STM32	STM32CubeIDE
IDE ESP32	Arduino IDE
Librairie Temps réel	FreeRTOS
Serveur Web	Flask (Python)
Interface Web	HTML / CSS / JS
Protocoles	UART / HTTP / WebSocket

2. Architecture générale du système

2.1. Présentation globale

Le système développé repose sur une architecture embarquée distribuée, composée de deux microcontrôleurs ayant des rôles distincts et complémentaires dans le contrôle et la supervision d'un robot mobile connecté.

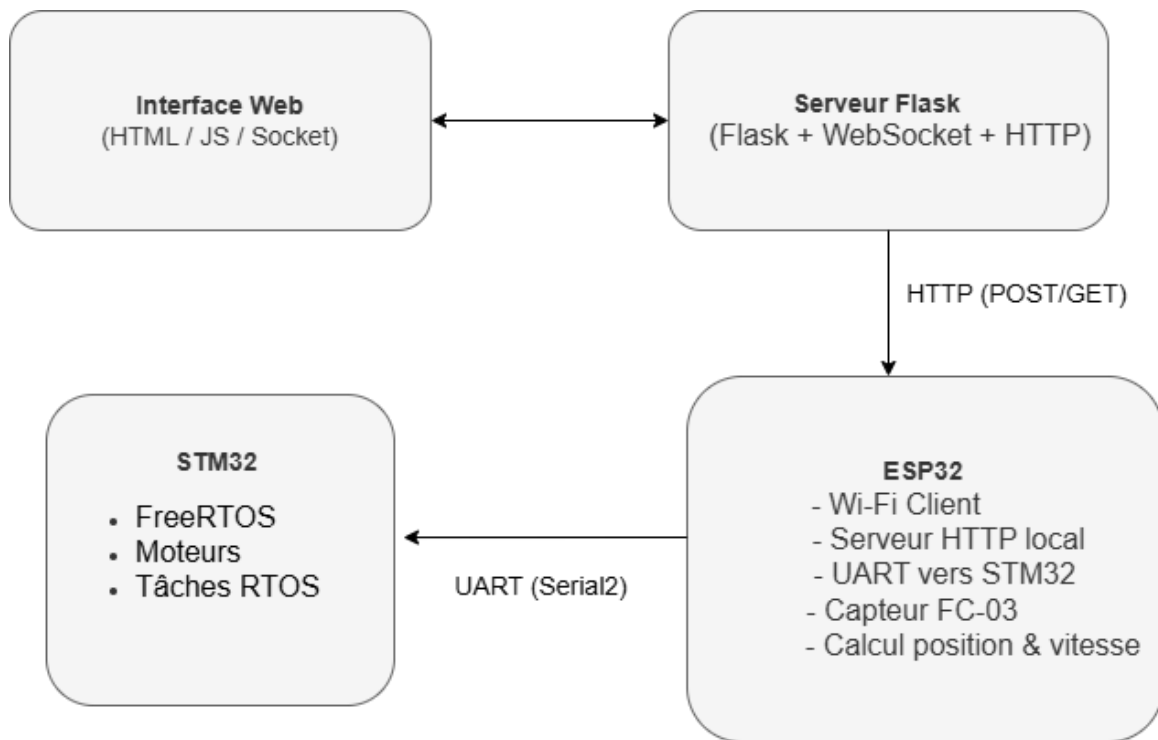
- Le *ESP32* est responsable de la connexion au réseau Wi-Fi, de l'hébergement d'un serveur HTTP, de la réception des commandes distantes, de la lecture du capteur de vitesse, du calcul de la position, et de l'envoi périodique de données au serveur Flask.
- Le *STM32*, quant à lui, est dédié à la gestion des moteurs du robot et à l'exécution des commandes via une architecture temps réel fondée sur FreeRTOS. Il reçoit les ordres depuis l'ESP32 via UART, les traite, et assure un mouvement fluide et précis du robot.

Ce découpage fonctionnel permet une séparation claire des responsabilités :

- Le ESP32 se concentre sur la communication réseau, l'acquisition de données et la logique applicative.
- Le STM32 gère le temps réel, le bas niveau matériel et la commande directe des actionneurs.

Cette organisation permet de garantir la réactivité et l'évolutivité du système, tout en assurant une bonne synchronisation entre les couches hautes (Web) et basses (moteurs).

2.2. Schéma d'architecture fonctionnelle



Le schéma présenté ci-dessus illustre l'organisation fonctionnelle de l'ensemble du système, en mettant en évidence les flux de données et les liens de communication entre les différents composants matériels et logiciels.

- **Interface Web utilisateur** : Située tout en haut de l'architecture, l'interface web permet à l'utilisateur de piloter le robot à distance. Développée en HTML/CSS/JavaScript, elle offre des boutons de commande (Avancer, Reculer, Stop, Gauche, Droite, etc.) et un affichage des données en temps réel comme la position, la vitesse ou l'orientation du robot. Cette interface est connectée en WebSocket au serveur Flask pour recevoir ces données dynamiquement.
- **Serveur Flask** (sur PC) : Le serveur Flask, développé en Python, agit comme intermédiaire entre l'utilisateur et le robot. Il sert à :
 - ❖ Recevoir les données envoyées par le robot via HTTP POST (position, vitesse, orientation)
 - ❖ Les transmettre en temps réel à l'interface utilisateur via WebSocket,
 - ❖ Relayer les commandes de l'utilisateur vers le robot, en les envoyant à l'ESP32 sous forme de requêtes HTTP GET.
- **ESP32** : L'ESP32 est un microcontrôleur Wi-Fi utilisé ici comme un cerveau communicant. Il est connecté au réseau Wi-Fi domestique et héberge un serveur HTTP local capable de recevoir des requêtes comme **/command?cmd=F**
- **STM32** : Le STM32 exécute les commandes envoyées par l'ESP32. Il est programmé sous FreeRTOS, un système d'exploitation temps réel qui permet de gérer plusieurs tâches en parallèle (ex. contrôle moteur, réception UART, gestion de sécurité...).

3. Fonctionnalités du système embarqué

3.1. Fonctionnalités côté ESP32

Le microcontrôleur ESP32 occupe une position centrale dans le système : il assure la communication réseau, la réception des commandes, la lecture des capteurs, le calcul de la position, et la transmission périodique des données vers le serveur web. Il sert d'interface entre l'utilisateur distant et le contrôleur STM32, en garantissant une circulation fluide et bidirectionnelle des informations.

3.1.1. Connexion Wi-Fi

Dès le démarrage, l'ESP32 tente de se connecter automatiquement au réseau Wi-Fi domestique configuré. Cette connexion est essentielle pour établir les communications HTTP avec le serveur Flask distant.

Le code embarqué utilise les bibliothèques WiFi.h pour la connexion, avec un contrôle sur le statut pour afficher ou relancer la connexion si nécessaire. Une fois connecté, l'adresse IP locale est affichée dans le terminal pour que le serveur puisse envoyer des commandes HTTP à l'ESP32.

3.1.2. Serveur HTTP local

L'ESP32 héberge un serveur HTTP léger accessible via son adresse IP locale. Ce serveur expose une route **/command**.

Le paramètre cmd est analysé pour détecter la commande transmise. Les commandes possibles sont :

- F : Avancer
- B : Reculer
- L : Tourner à gauche
- R : Tourner à droite
- S : Stop
- RESET : Réinitialisation du robot

Chaque commande reçue est aussitôt transmise au STM32 via **UART** pour exécution.

3.1.3. Communication UART avec le STM32

La communication entre l'ESP32 et le STM32 s'effectue via UART sur les broches **GPIO 16 (TX)** et **GPIO 17 (RX)**. L'ESP32 joue ici le rôle de maître, envoyant des commandes sous forme de caractères simples (F\n, S\n, etc.).

Il écoute également les messages en retour depuis le STM32 (par exemple un retour d'état ou de confirmation), ligne par ligne jusqu'au caractère de fin.

Cette communication est synchrone et essentielle, car elle permet au STM32 d'agir sur les moteurs selon les commandes reçues.

3.1.4. Lecture du capteur de vitesse (FC-03)

Un capteur optique FC-03 est connecté à l'ESP32 sur la broche GPIO 4, configurée en interruption externe. Le capteur détecte les impulsions générées par un disque percé fixé sur la roue du robot.

Chaque fois qu'un trou passe devant la fourche du capteur, une interruption est déclenchée, incrémentant un compteur d'impulsions.

En utilisant la formule :

$$\text{vitesse} = (\text{impulsions} / \text{temps}) \times (\text{circonférence roue} / \text{nombre de trous})$$

3.1.5. Calcul de la position (X, Y)

Grâce aux mesures de vitesse et à l'orientation du robot, l'ESP32 effectue un suivi de la position en temps réel.

En supposant un mouvement à vitesse constante sur un intervalle court Δt , la position est mise à jour par :

$$\text{posX} += \text{distance} \times \cos(\text{angle_en_radians})$$

$$\text{posY} += \text{distance} \times \sin(\text{angle_en_radians})$$

L'angle d'orientation (orientationDeg) peut être maintenu à jour selon la dernière commande envoyée (par exemple : tourner gauche/droite ajuste $\pm 90^\circ$).

3.1.6. Envoi des données vers le serveur Flask

Toutes les 2 secondes, l'ESP32 envoie un paquet JSON au serveur Flask via une requête HTTP POST sur l'endpoint /update.

Ce paquet contient :

- La position actuelle {"x": ..., "y": ...}
- L'orientation {"angle": ...}
- La vitesse actuelle {"speed": ...}

Le format JSON est interprété par Flask, qui redistribue les données via WebSocket à l'interface utilisateur. Cela permet un affichage en temps réel du mouvement du robot sur le navigateur.

3.2. Fonctionnalités côté STM32

Le STM32F429I joue le rôle de contrôleur moteur temps réel dans notre système robotique. Il exécute les commandes reçues via UART depuis l'ESP32, en activant ou désactivant les moteurs du robot selon l'instruction correspondante. Grâce à l'intégration du système d'exploitation temps réel FreeRTOS, les différentes fonctions du STM32 sont organisées sous forme de tâches multitâches concurrentes, assurant ainsi une exécution fluide et réactive, même en présence de multiples événements.

3.2.1. Réception UART des commandes

Le STM32 reçoit les commandes de déplacement (F, B, L, R, S, RESET) depuis l'ESP32 via l'interface UART1. La configuration est faite pour utiliser les interruptions UART, assurant une détection rapide et non bloquante des messages entrants.

À chaque réception d'un caractère (ou d'une chaîne terminée par \n), une tâche dédiée analyse la commande et la stocke dans une variable partagée, qui est ensuite utilisée par la tâche de contrôle moteur.

3.2.2. Contrôle des moteurs via L298N

Le STM32 pilote les moteurs à travers un driver L298N, en utilisant :

- Des broches **GPIO** (PD0 à PD3) pour les signaux de direction (IN1 à IN4)
- Des broches **PWM** (PD4 et PD5) pour la modulation de vitesse (ENA et ENB)

Selon la commande reçue (F pour avancer, L pour tourner, etc.), le STM32 active les broches correspondantes pour générer le mouvement souhaité. Une fonction de sécurité permet de stopper les moteurs (S) à tout moment.

3.2.3. Structure des tâches FreeRTOS

Le programme STM32 utilise **FreeRTOS** pour organiser ses opérations de manière multitâche. Voici un exemple de structure typique :

TÂCHE	FONCTION	PRIORITÉ
TASK_UART_RECEPTION		Haute

	Lire les messages UART, les interpréter et mettre à jour l'état des commandes	
TASK_MOTOR_CONTROL	Lire l'état actuel de la commande et activer les moteurs en conséquence	Normale
TASK_SUPERVISION	Afficher l'état du système sur un écran LCD ou via UART	Basse

Chaque tâche est définie comme une boucle infinie contrôlée par FreeRTOS, avec des mécanismes de sémaphores ou queues si besoin pour synchroniser les données entre tâches.

3.2.4. Gestion des interruptions

Le système utilise des interruptions matérielles, en particulier :

- **USART1_RX** pour lire les données entrantes sans bloquer le reste du programme
- D'autres interruptions peuvent être envisagées pour surveiller des capteurs ou gérer des erreurs moteurs

Les priorités des interruptions sont soigneusement définies pour garantir la réactivité du système sans provoquer de conflit entre tâches.

3.2.5. Envoi des données vers l'ESP32

Bien que le STM32 soit principalement récepteur, il peut également envoyer des messages d'état ou de confirmation vers l'ESP32, par exemple :

- "OK" après exécution d'une commande
- "Error" si un problème est détecté
- Informations sur l'état des moteurs

Ce dialogue bidirectionnel par UART permet à l'ESP32 d'ajuster son comportement ou d'alerter l'utilisateur via l'interface Web.

4. Serveur Flask et Interface Web

4.1. Serveur Flask

Le serveur Flask, développé en Python, sert d'intermédiaire entre le robot et l'utilisateur. Il a deux fonctions principales :

- Recevoir les données du robot : toutes les 2 secondes, l'ESP32 envoie sa position (x, y), sa vitesse, et son orientation au serveur, via une requête HTTP POST sur /update.

- Transmettre ces données à l'interface web en temps réel grâce à WebSocket (avec Flask-SocketIO). Dès qu'une nouvelle donnée arrive, elle est envoyée automatiquement au navigateur.

4.2. Interface Web

L'interface web, accessible depuis un navigateur, permet à l'utilisateur de :

- **Piloter le robot** grâce à des boutons (Avancer, Reculer, Gauche, Droite, Stop). Chaque clic envoie une commande à l'ESP32 via une requête HTTP GET.
- **Voir l'état du robot en direct** : la page affiche les données reçues (position, vitesse, orientation) grâce à une connexion WebSocket, sans avoir besoin de recharger la page.

5. Communication entre les composants

La communication est un élément essentiel du système. Elle permet de faire circuler les commandes et les données entre l'interface utilisateur, l'ESP32 et le STM32. Deux types de communication sont utilisés : UART (liaison série) et HTTP/WebSocket (réseau).

5.1. Communication ESP32 ↔ STM32 (UART)

L'ESP32 et le STM32 sont connectés via une liaison UART. C'est un protocole simple et rapide pour échanger des données sérieusement.

- Dans un sens : l'ESP32 envoie les commandes (F, B, L, R, S, RESET) au STM32.
- Dans l'autre sens : le STM32 peut envoyer un message de confirmation ou d'état.

Les données sont envoyées caractère par caractère ou ligne par ligne, avec un retour à la ligne \n pour marquer la fin de message. L'ESP32 utilise Serial2 pour envoyer et recevoir, et le STM32 utilise des interruptions UART pour ne rien manquer.

5.2. Communication ESP32 ↔ Serveur Flask (HTTP)

La communication entre l'ESP32 et le serveur Flask se fait via le réseau Wi-Fi en utilisant le protocole HTTP.

- Pour recevoir les commandes : le serveur Flask envoie une requête HTTP GET à l'ESP32 (exemple : /command?cmd=F).
- Pour envoyer les données : l'ESP32 envoie une requête HTTP POST au serveur Flask (toutes les 2 secondes), avec un fichier JSON contenant sa position, sa vitesse et son orientation.

Cette communication permet à l'ESP32 de rester connecté à distance et de rendre le robot visible en temps réel sur l'interface.

5.3. Communication Serveur Flask ↔ Interface Web (WebSocket)

Pour que l'utilisateur voie les données du robot en temps réel, le serveur Flask utilise WebSocket, un protocole qui permet de pousser des données vers le navigateur dès qu'elles arrivent (sans rechargement).

Dès que le serveur reçoit une mise à jour du robot, il l'envoie automatiquement à l'interface web :

```
socketio.emit('robot_data', data)
```

6. Conclusion

Ce projet a permis la mise en œuvre complète d'un robot mobile communicant, basé sur une architecture embarquée distribuée entre un ESP32 et un STM32. Il a démontré l'intérêt d'associer la connectivité réseau à un pilotage temps réel, en intégrant plusieurs domaines clés de l'ingénierie embarquée : communication série (UART), traitement multitâche (FreeRTOS), interface web, et supervision à distance.

Le ESP32 a assuré la gestion du Wi-Fi, le traitement des commandes HTTP, la lecture du capteur de vitesse FC-03, le calcul de la position et l'envoi des données vers un serveur Flask. De son côté, le STM32 a exécuté les ordres moteurs sous FreeRTOS, en assurant une réponse rapide et stable, même en cas de sollicitations fréquentes.

Cette séparation des responsabilités entre les deux microcontrôleurs a permis d'obtenir un système à la fois réactif, modulaire et évolutif. L'interface web, bien que simple, a permis à l'utilisateur de piloter le robot à distance et de suivre son état en temps réel, montrant ainsi la faisabilité d'un système IoT embarqué pilotable via navigateur.

En plus des compétences techniques acquises, ce projet a mis en évidence l'importance :

- de la robustesse des communications,
- de la gestion des tâches concurrentes,
- et de la cohérence globale entre les composants logiciels et matériels.

Les améliorations futures pourraient porter sur l'ajout de capteurs plus avancés, une interface web enrichie, ou encore l'intégration dans un système multi-robots.