RAPPORT DE PROJET

Système Intelligent de Surveillance

Cours : CEG 4566/SEG 4545/CSI 4541

Professeur : Dr. Mohamed N. Rahmani

Travail par :

Maxime Couture, 300236983, mcout088@uottawa.ca

Nathan Gawargy, 300232268, ngawa073@uottawa.ca

Rémi Gros-Jean, 300258238, rgros044@uottawa.ca

Syed Yead Zaman, 300245018, szama023@uottawa.ca

Numéro de groupe : 9

Date de soumission : 3 avril 2025

Université d'Ottawa

Hiver 2025

TABLE DES MATIÈRES

TABLE DES FIGURES

TABLE DES TABLEAUX

INTRODUCTION

Le projet vise à développer un système de surveillance et de détection portable et réactif, capable de scanner l'environnement en continu et d'alerter l'utilisateur en temps réel en cas de détection qui dépasse la zone de proximité. Ce système repose sur des technologies embarquées et de la logique optimisée pour garantir une détection rapide et fiable. Le système est aussi conçu pour être compact et économe en énergie, ce qui lui rend adaptable à divers scénarios d'application. Les informations capturées par les capteurs sont transmises à une application web via une communication Bluetooth, permettant ainsi à l'utilisateur de visualiser les données en temps réel. Cette interface intuitive permet une surveillance efficace, simplifiant l'analyse des alertes et facilitant une prise de décision rapide pour l'utilisateur. L'application web permet également de configurer le système à distance via une connexion Bluetooth, offrant à l'utilisateur la possibilité de personnaliser le système de surveillance.

CONCEPTION

Exigences

Cette section présente les exigences du projet, qui définissent le fonctionnement du système et servent comme des critères de succès pour le produit final.

Exigences Fonctionnelles

- **Détection des objets :** Le système doit détecter les objets présents dans son environnement dans la plage de détection définie.
- **Alarme de proximité** : Lorsqu'un objet entre dans une zone de proximité définie par l'utilisateur, une alarme sonore doit se déclencher pour signaler la présence et potentiellement dissuader l'intrusion.
- **Détection en temps réel** : Le système doit analyser son environnement et signaler les détections sans délai notable.
- **Balayage continu** : Le capteur de détection doit scanner en permanence son environnement dans sa plage de détection pour identifier toute présence d'objet.
- **Transmission des données** : Les données collectées doivent être envoyées à une application web avec de la communication sans fil.
- **Visualisation des détections** : L'application web doit présenter les données de détection et leur proximité pour l'utilisateur.
- **Gestion de l'énergie** : Lorsqu'aucun mouvement n'est détecté, le système doit passer en mode basse consommation pour préserver l'énergie.
- **Configuration du système à distance** : L'application web permet l'utilisateur de configurer le système à distance avec une connexion sans fil.

Exigences Non-Fonctionnelles

- **Fiabilité :** Les données sur les détections et leurs proximités doivent être juste et assez précis.
- **Portabilité :** Le système doit être portable pour être utilisable dans différents scénarios.
- **Latence faible :** Le délai entre une détection et la visualisation sur l'application web ne devrait pas dépasser 1 seconde.

Architecture Haut-Niveau

Le diagramme de conception est présenté dans la Figure 1. Il a été réalisé à l'aide du langage de modélisation architecturale ArchiMate [1], qui permet de représenter des systèmes en utilisant différents modules et de décrire les relations entre les composants. Une légende est incluse dans le diagramme afin d'expliquer les différents types de module et de relation.

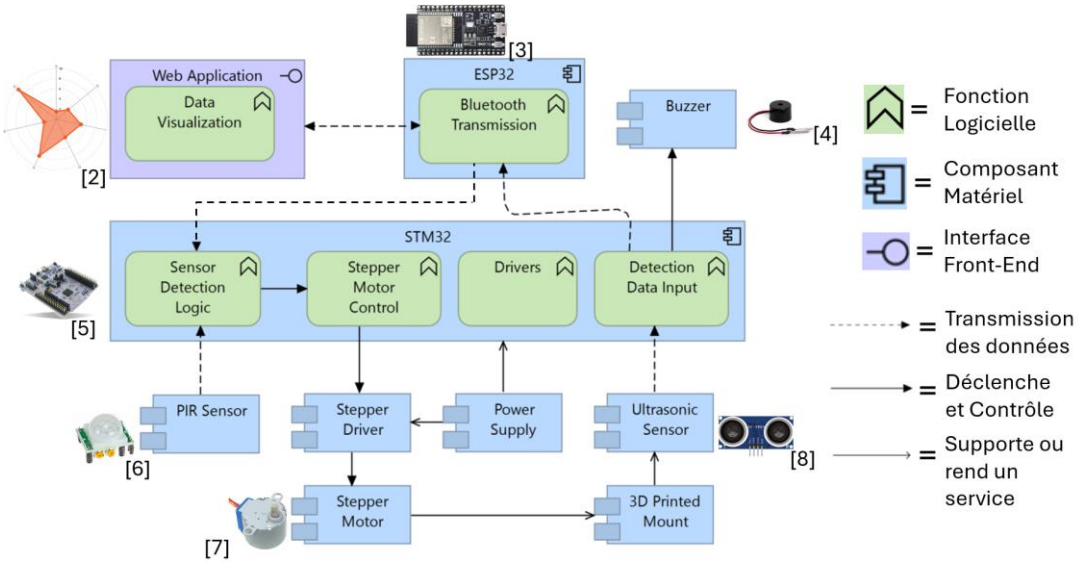

Figure 1 : *Conception Architecturale du système*

Description des fonctions logicielles

La carte STM32 est responsable de toute la logique de contrôle et agit comme contrôleur principal du système. Le code s'exécutant en continu sur la carte gère plusieurs fonctions essentielles :

- **Sensor Detection Logic :** Code qui gère l'activation du système en fonction des entrées du capteur PIR. Si aucun mouvement n'est détecté pendant un certain délai, le moteur s'arrête et le capteur ultrasonique se désactive. Dès qu'un mouvement est détecté par le capteur PIR, le système redémarre automatiquement. Reçoit les configurations du système choisi par l'utilisateur sur l'application web et met à jour le système selon ces paramètres.
- **Stepper Motor Control :** Code responsable pour contrôler le stepper driver en fonction des commandes définies par la logique de détection.
- **Drivers** : Bibliothèques nécessaires pour interfacer la carte STM32 avec les différents composants matériels.

- **Detection Data Input :** Réception des mesures du capteur ultrasonique (distance des objets détectés) et les informations du moteur (angle de rotation). Ces informations sont transmises au ESP32 qui s'occupe de la communication avec l'application web. Il faut aussi avoir de la logique supplémentaire pour déclencher l'alarme lorsqu'il a un objet qui entre dans une zone de proximité proche du capteur.

Les données sont transmises en série du STM32 vers la carte ESP32. Ensuite, un programme qui s'exécute sur le ESP32 prend en charge la transmission de ces données vers l'application web avec la communication Bluetooth. L'application web est constituée d'un programme qui se connecte au canal Bluetooth pour recevoir les données du système de surveillance et affiche les données à l'aide d'une représentation visuelle. L'application web comprend également des paramètres configurables sur l'interface, permettant à l'utilisateur d'ajuster le système à distance selon ses besoins.

Description des composants matériels

Le tableau ci-dessous fourni les détails sur les composants matériels nécessaires du projet.

Tableau 1 : *Liste des composants du système*

Composant	Description
Microcontrôleur STM32F401RE	Microcontrôleur principal du système, responsable de la gestion des capteurs, du moteur et de la communication avec les autres composants. Il exécute la logique de détection et contrôle l'ensemble du processus en temps réel.
Capteur ultrasonique HC-SR04	Utilisé pour mesurer la distance des objets détectés dans l'environnement. Il est monté sur le moteur pour scanner différentes directions et fournir des données précises sur la présence d'obstacles.
Capteur PIR	Détecte les mouvements à proximité et active le système en mode normal lorsqu'un mouvement est perçu. Si aucun mouvement n'est détecté après un certain délai, il met le système en mode veille.
Pilote de moteur (stepper driver ULN2003)	Module électronique permettant de contrôler le moteur. Il reçoit des signaux de commande pour ajuster la rotation et la vitesse du moteur.
Moteur (stepper motor 28BYJ-48)	Moteur pour faire tourner le capteur ultrasonique, lui permettant de scanner l'environnement à différents angles.
ESP32	Utilisé pour communiquer avec l'application web via le module Bluetooth qui est intégré dans la carte.
Buzzer	Génère une alarme sonore lorsqu'un objet est détecté dans une zone de proximité définie par l'utilisateur, servant ainsi d'alerte pour l'utilisateur et de dissuasion pour les intrus.
Plateforme et Support 3D	Support imprimé en 3D utilisé pour fixer le capteur ultrasonique au moteur. Ce support permet une fixation stable pendant le mouvement de rotation du moteur. Une plateforme a aussi été utilisé pour avoir un système plus compact.
Source d'alimentation	Source externe d'alimentation qui fournit un voltage de 5V à la carte STM32 et à la carte ESP32.
Module d'alimentation pour le pilote de moteur	Fournit l'énergie nécessaire au fonctionnement du pilote de moteur et du moteur pas-à-pas.

IMPLÉMENTATION

Implémentation du matériel

Le matériel a été configuré selon le schéma présenté ci-dessous *(voir Figure 2)*. Tous les composants sont connectés directement au STM32 qui fournit l'alimentation à ceux-ci, à l'exception du pilote de moteur (stepper driver) et la carte ESP32. Le stepper driver nécessite une alimentation supplémentaire pour fournir la puissance et la tension nécessaires à son fonctionnement, et le ESP32 est connecté directement à une source d'alimentation externe.

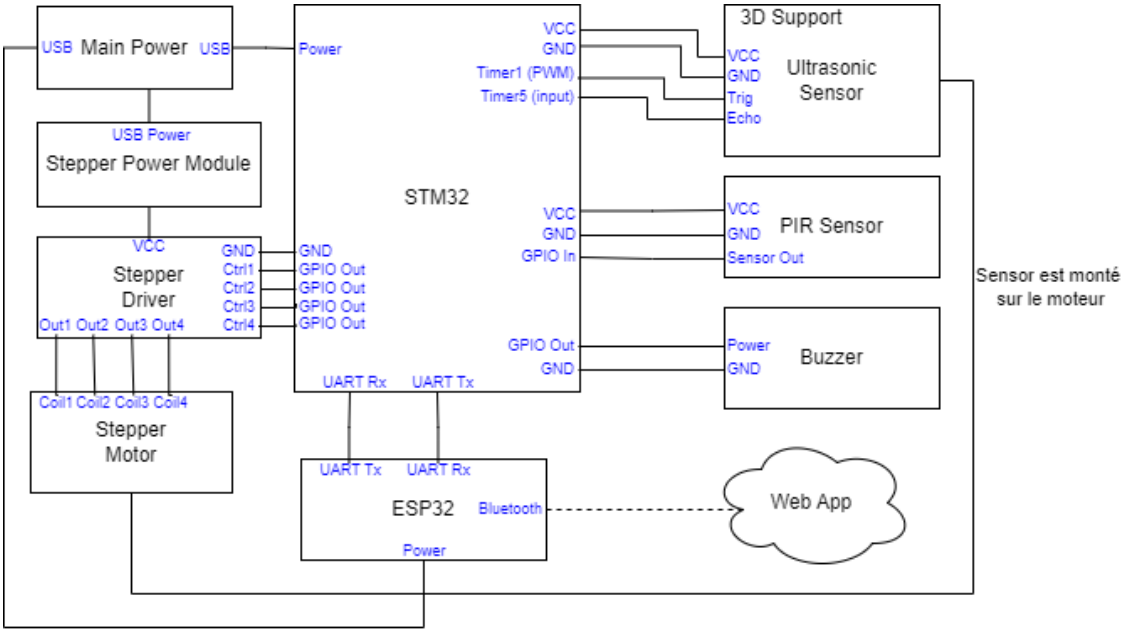

Figure 2 : *Connexions et communication entre composants matériels*

Le système de sécurité est construit autour du microcontrôleur STM32, sélectionné pour ses puissantes capacités de traitement et sa gestion efficace des tâches en temps réel. Celui-ci sert de l'unité centrale, gérant la logique de détection en temps réel et contrôlant l'ensemble des opérations des capteurs et actionneurs. Bien que la conception de systèmes avec le STM32 soit plus complexe et plus longue, elle offre de meilleures performances et un contrôle matériel plus flexible que d'autres microcontrôleurs, comme l'Arduino.

Un capteur ultrasonique, fixé sur un moteur grâce à un support imprimé en 3D, effectue un balayage précis de l'environnement afin de mesurer les distances par rapport aux obstacles. Le moteur, piloté par un module dédié, assure une rotation contrôlée du capteur dans une zone et règlementation précis. Un buzzer émet un signal sonore lorsqu'un objet est détecté à une distance prédéfinie par le capteur ultrasonique. Pour économiser de l'électricité, le système utilise un capteur infrarouge passif (PIR) qui détecte les mouvements à proximité. Après une période avec aucunes détections et aucun mouvement, le système passe en mode veille afin de réduire sa consommation énergétique.

La communication des données en temps réel est assurée par une carte ESP32, qui reçoit les données du STM32 en série (UART) et transmet les mesures du capteur à une application web. La connectivité Bluetooth a été choisie pour son efficacité énergétique et sa facilité d'intégration. L'utilisation des minuteries fourni par la carte STM32 était essentielle pour assurer le bon fonctionnement du capteur ultrasonique. Le *Timer1* a été configuré en mode génération PWM (Pulse Width Modulation) afin d'émettre une impulsion servant à déclencher la mesure de distance du capteur. Le *Timer5* a été utilisé en mode capture d'entrée pour mesurer la durée de l'impulsion de retour (écho), permettant ainsi de calculer la distance des objets détectés *(voir Figure 7)*.

Le matériel a été connecté et implémenté selon le diagramme présenté dans la Figure 1. Les images ci-dessous présente l'implémentation actuelle des différents composants matériels ensemble pour former le système *(voir Figure 3 et 4)*.

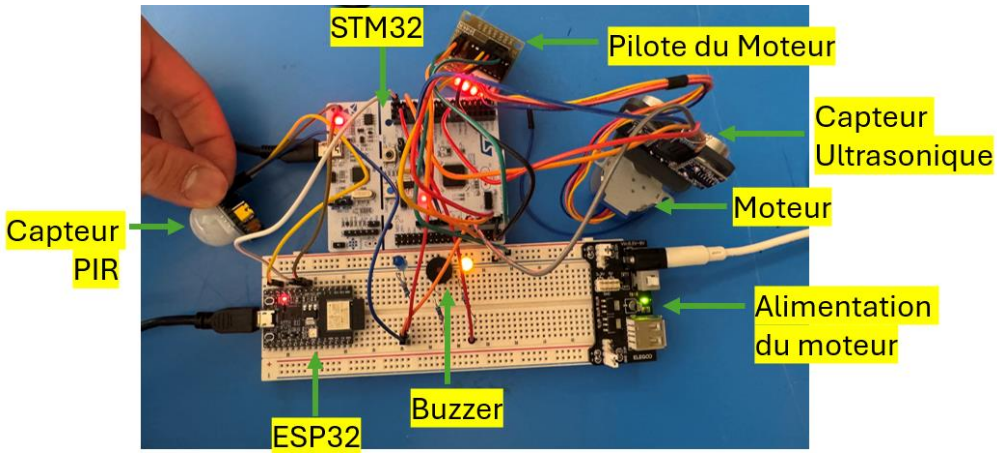

Figure 3 : *Configuration et connexion des composants matériels*

La Figure 4 présente le système final où le capteur ultrasonique et le moteur sont fixés sur un support imprimé en 3D. Un trou dans le boîtier est dédié pour le capteur PIR pour permettre la détection de mouvement, tandis que tous les autres fils et composants matériels sont logés dans le boîtier. Le boîtier a été choisi et conçu avec des couleurs qui sont à la fois esthétiques et fonctionnelles, et permet de stocker tous les composants matériels du système.

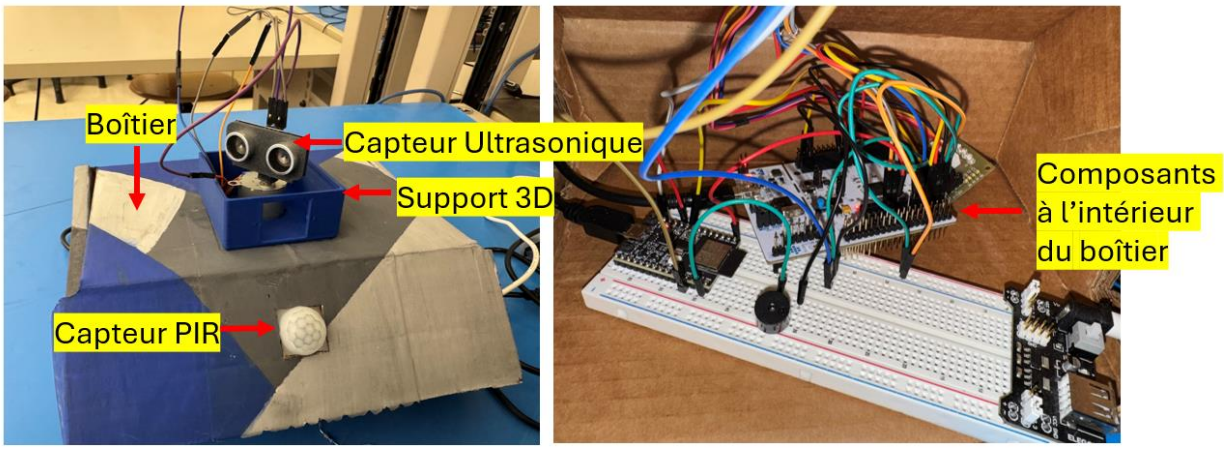

Figure 4 : *Configuration final du système avec le boîtier et le support 3D*

Implémentation du logiciel

L'image ci-dessous présente les différents modules qui exécutent du logiciel *(voir Figure 5)*. L'environnement de développement STM32CubeIDE a été utilisé pour le développement du code s'exécutant sur la carte STM32. Ceci a facilité l'implémentation du logiciel afin que plusieurs librairies soient fournies et peuvent être utilisées. Les drivers inclus par défaut par le projet STM32 ainsi que les fichiers comme le *syscalls.c* et *stm32f4xx_it.c* sont essentiels pour permettre la configuration et l'interfaçage avec le matériel et les fonctionnalités bas-niveau de la carte STM32. Cependant, d'autres fichiers ont été développé par les membres de ce groupe pour permettre un contrôle de différents composants matériels, notamment pour le capteur ultrasonique et le pilote du moteur (stepper driver). Les différents modules de logiciels principaux sont détaillés dans les sections suivantes.

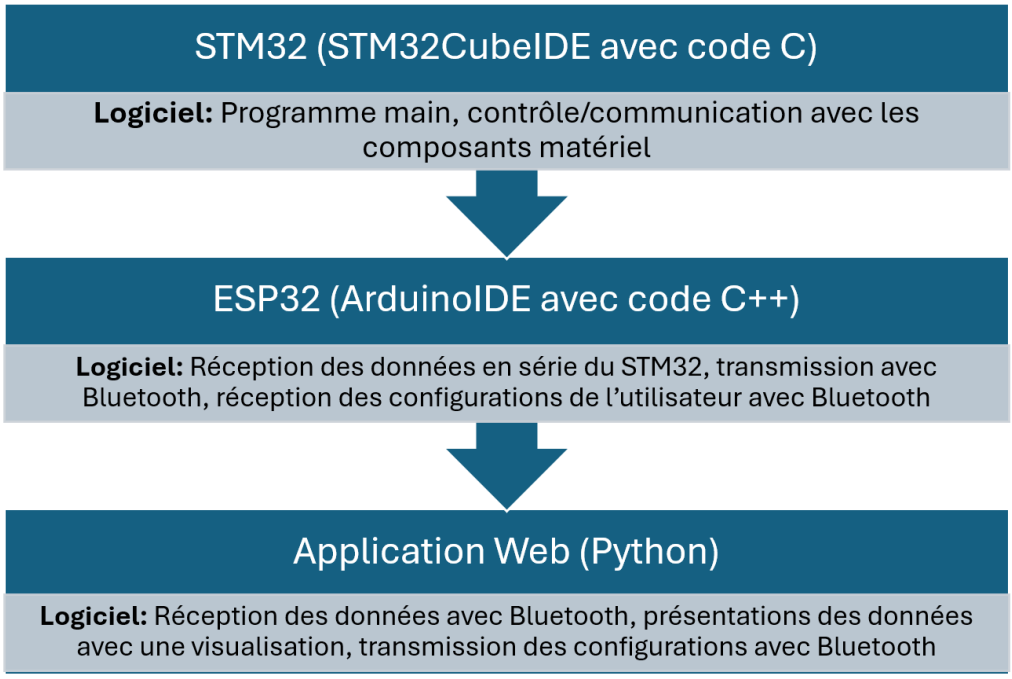

Figure 5 : *Hiérarchie des différents composants qui exécutent du code logiciel*

Les données sont partagées entre le STM32 et le ESP32 via une communication UART, tandis que la communication Bluetooth est utilisée pour l'échange de données entre le ESP32 et l'application web. Le ESP32 joue le rôle d'intermédiaire entre le STM32 et l'application web, facilitant ainsi une communication sans fil entre ces deux modules. La séquence de transmission des données entre les composants du système est présentée dans la Figure 6.

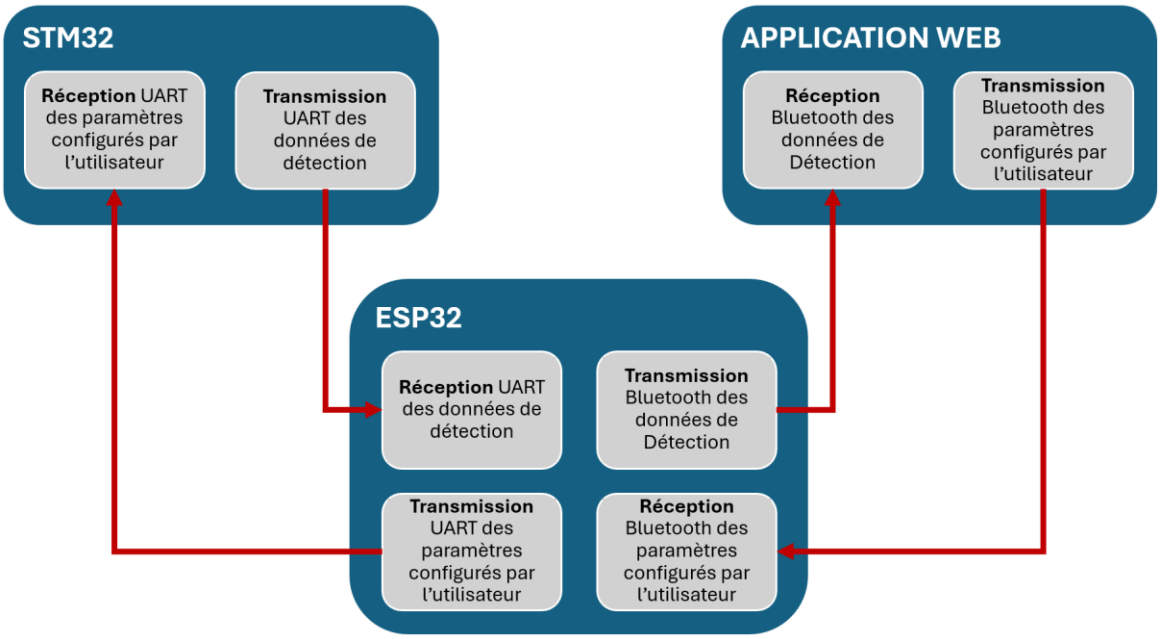

Figure 6 : *Communication entre les composants du système pour le partage des données*

<u>Contrôle du capteur ultrasonique STM32 (ultrasonic.c)</u>

Ce fichier contient l'implémentation des fonctions permettant de contrôler et de recevoir les données du capteur ultrasonique. Il comprend des fonctions responsables de l'initialisation du capteur ainsi que de la conversion des données du capteur en distances quantifiables (en centimètres). L'obtention de la distance des objets se fait en déclenchant un pulse ultrasonique, puis en attendant la fin de la mesure du signal de retour (écho). La Figure 7 démontre la méthode utilisée pour détecter des objets et pour l'obtention d'une mesure de la distance de ceux-ci. L'équation de la distance se sert d'une multiplication par la valeur de la vitesse du son (0.034 cm/µs dans une chambre de 20°C). La division par 2 est nécessaire car l'écho parcourt la distance jusqu'à l'objet puis revient au capteur. Étant donné que l'on souhaite uniquement mesurer la distance entre le capteur et l'objet, il est donc nécessaire de diviser par 2 pour obtenir la distance à sens unique. Cette distance calculée est ensuite validée par rapport aux valeurs minimales et maximales prédéfinies. Si la distance est dans la plage valide, elle est retournée ; sinon, une valeur de -1 est retournée pour indiquer une mesure invalide *(voir Figure 15 dans l'Annexe pour le code)*.

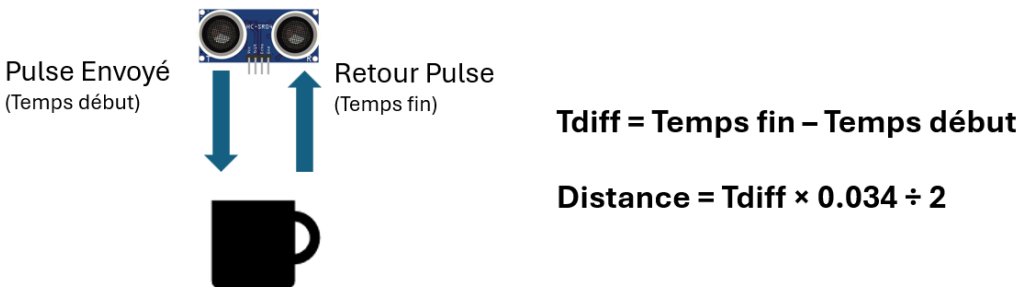

Figure 7 : *Illustration du calcul de la distance pour le capteur ultrasonique*

Contrôle du moteur STM32 (stepper.c)

Ce code permet de piloter un moteur avec la possibilité de changer la direction et de contrôler la vitesse du moteur. Il inclut des fonctions pour activer et désactiver le moteur, permettant ainsi un contrôle précis du mouvement du moteur. Il a aussi une fonction qui permet d'obtenir l'angle courant du moteur *(voir Figure 16 et 17 dans l'Annexe pour le code)*.

Programme principale STM32 (main.c)

Le programme principal est responsable de l'initialisation des différents périphériques et de la configuration des broches utilisées pour connecter les composants matériels à la carte STM32. Le noyau FreeRTOS est utilisé pour lancer la tâche *ultrasonicTask*, qui gère la plupart de la logique du système *(voir Figure 18 et 19 de l'Annexe pour le code)*. Cette tâche effectue les opérations suivantes :

- Mesure la distance à l'aide du capteur ultrasonique (utilise les fonctions de *ultrasonic.c*).
- Récupère l'angle actuel du moteur stepper (utilise les fonctions de *stepper.c*)
- Transmet les données de distance et d'angle via UART vers le ESP32
- Contrôle la direction du moteur pas à pas et l'active/désactive en fonction de l'angle (utilise les fonctions de *stepper.c*)
- Déclenche l'alarme si un objet est trop proche
- Gère la planification du *'sleep mode'* en fonction de la distance mesurée et de l'état du capteur de mouvement PIR.

Le diagramme ASM illustré dans la Figure 8 représente la séquence des événements et du contrôle du programme principal. Il est important de noter que cette tâche s'exécute en continu, afin que les systèmes embarqués en temps réel fonctionnent de manière cyclique et ininterrompue. Des détections qui sont 30 cm ou plus proches sont considérés comme des intrusions au système et déclenche l'alarme. Cette distance peut être configuré dans le système par l'utilisateur.

Une autre fonction de type Callback a été conçu, qui est automatiquement appelée lorsqu'une interruption se produit *(voir Figure 20 et 21 de l'Annexe pour le code)*. Cette fonction est déclenchée lors d'une interruption de réception UART, signalant que les données de l'application web ont été correctement reçues par le ESP32 et les données sont prêts à être transmis au STM32. Ces données correspondent aux configurations sélectionnées par l'utilisateur sur l'interface web. Elles sont transmises au ESP32 via la connexion Bluetooth, puis envoyées au STM32 par la communication UART.

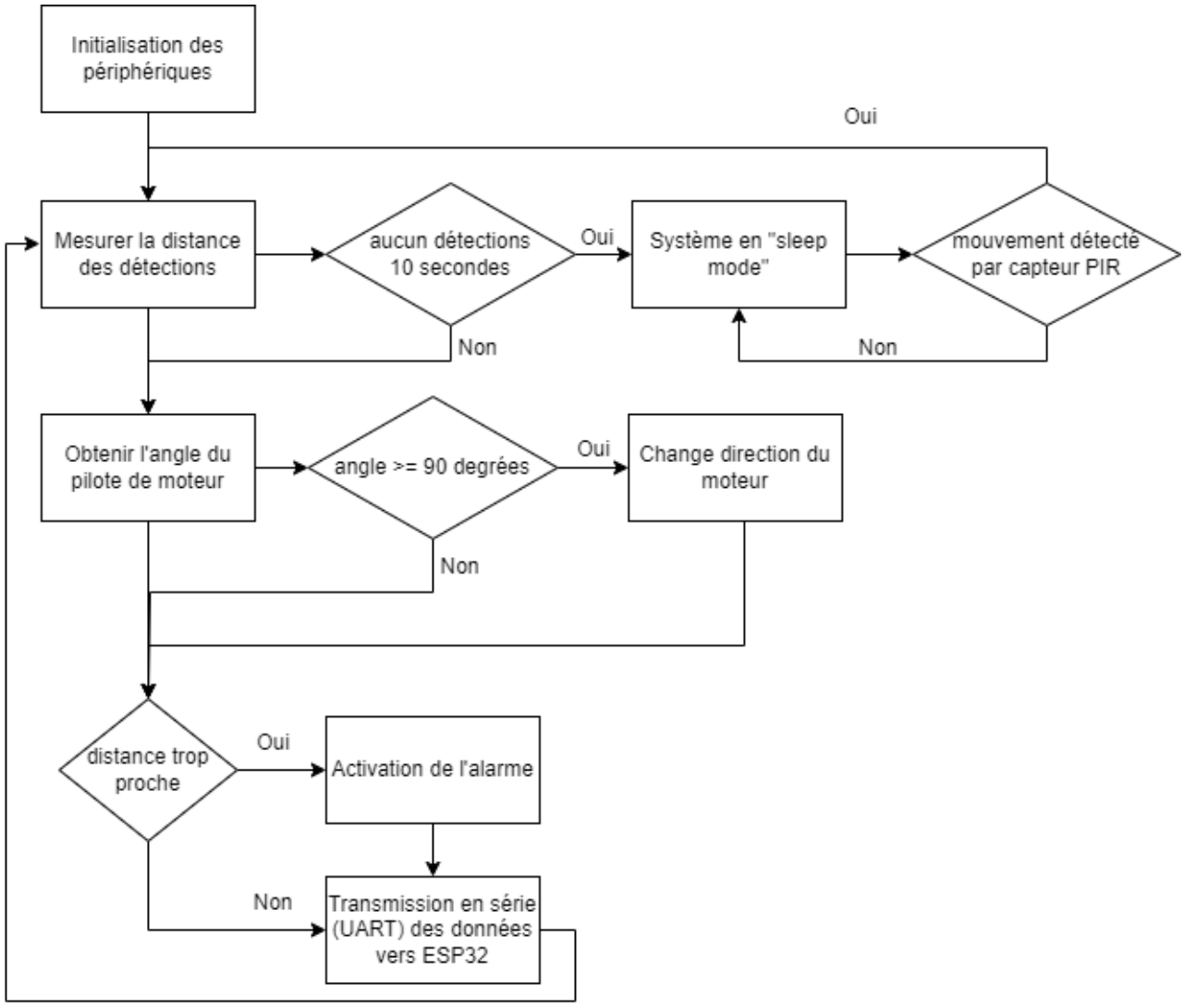

Figure 8 : *Diagramme ASM illustrant la séquence d'évènement du programme principale*

<u>Communication Bluetooth avec le ESP32</u>

Les données du programme principale roulant sur la carte STM32 sont envoyés avec la communication UART à la carte ESP32. Du code en langage C++ a été implémenté pour configurer la connexion Bluetooth, de recevoir les données en série et puis de les envoyés à l'application web *(voir Figure 22 et 23 de l'Annexe pour le code)*. De plus, une seconde connexion Bluetooth est nécessaire pour recevoir les paramètres sélectionnés par l'utilisateur via l'application web, puis les transmettre au STM32 via la communication UART *(voir Figure 24 de l'annexe pour le code)*.

Application Web

La dernière partie du logiciel repose sur une application Python qui reçoit les données à travers connexion Bluetooth. La bibliothèque Dash a été utilisée pour créer une application interactive facilitant la visualisation des données [9], et la bibliothèque Bleak [10] assure la communication en Bluetooth Low Energy (BLE). Les données de détection reçues par l'application sont représentées sous forme d'une visualisation radar, affichant à la fois la distance et l'angle de détection *(voir la Figure 25 de l'Annexe pour le code)*. L'interface intègre également les paramètres configurables du système, présentés avec différentes options adaptées à chaque réglage. Les paramètres configurables du système sont les suivants :

- **Vitesse du moteur :** Définit la vitesse du moteur, influençant la fréquence des détections.
- **Distance maximale de détection :** Configure la portée maximale du capteur ultrasonique pour la détection d'obstacles.
- **Seuil de détection d'alarme :** Distance minimale à laquelle une détection déclenche l'alarme.
- **Timeout du mode veille :** Spécifie le délai avant que le système passe en mode veille lorsqu'il n'a aucune détection.

RÉSULTATS

Des images et captures d'écrans sont présentés dans cette section qui démontre le fonctionnement du système implémenté. Une fenêtre du terminal est aussi incluse dans la plupart des images qui présente les données de détection qui seront affichés sur l'application web et des messages de statut du système. Dans la Figure 9, un objet se trouve dans la plage de détection qui active l'alarme. Dans ce cas, une alerte est affichée en temps réel sur l'application web, et le buzzer se déclenche.

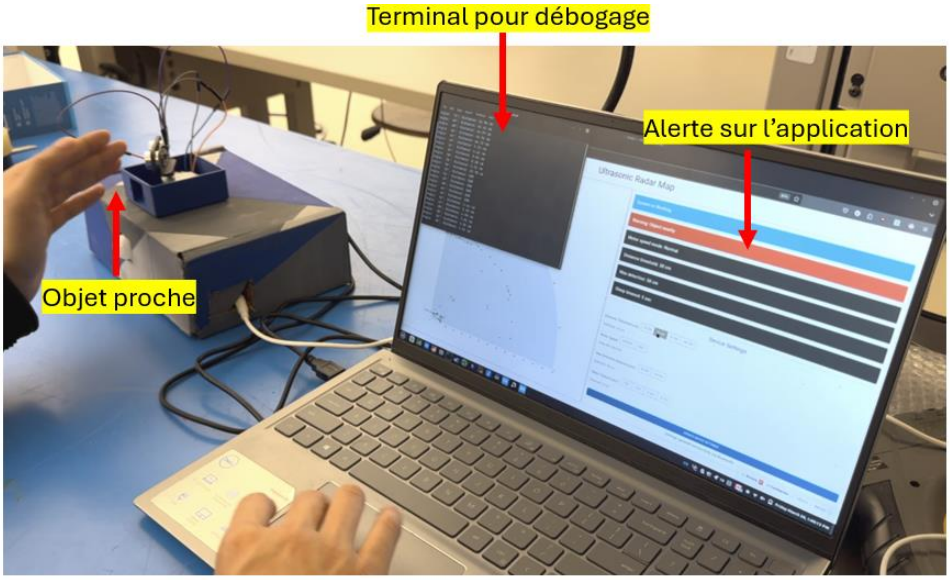

Figure 9 : *Détection des objets et déclenchement de l'alarme*

Lorsqu'aucune détection n'est détectée pendant un intervalle de temps défini (ici, 2 secondes), le mode veille s'active *(voir Figure 10)*. L'application web indique alors que les détections sont *'out of range'*, ce qui signifie qu'aucun objet n'est détecté par le capteur à ultrasons. De plus, le statut du système affiche que celui-ci est en mode veille *(System is sleeping)*. Dans ce mode, le moteur et le capteur ultrasonique s'éteint afin de conserver de l'énergie.

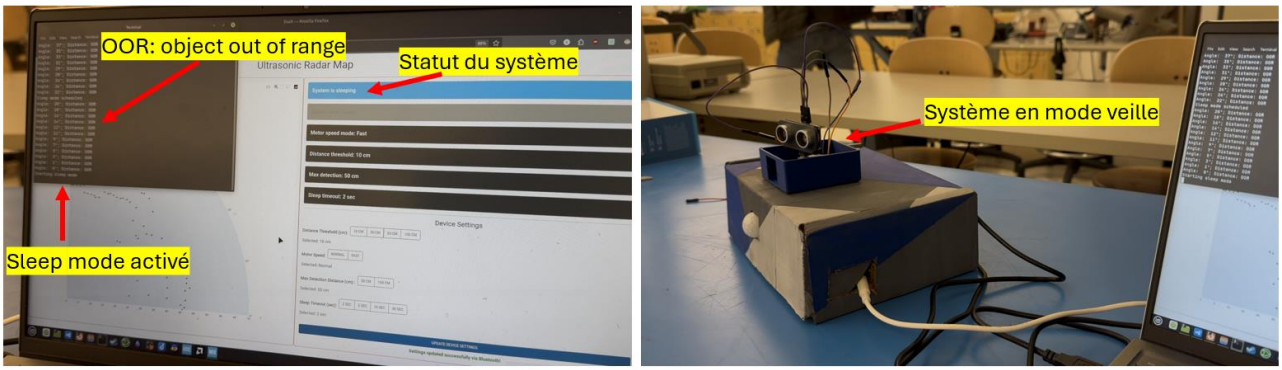

Figure 10 : *Activation du mode veille lorsqu'il n'a aucune détection*

Dès qu'un mouvement est détecté par le capteur PIR, le système se réactive et son statut est mis à jour sur l'interface web *(voir Figure 11)*. Le système reprend le fonctionnement normal en détectant les objets en mouvement continu.

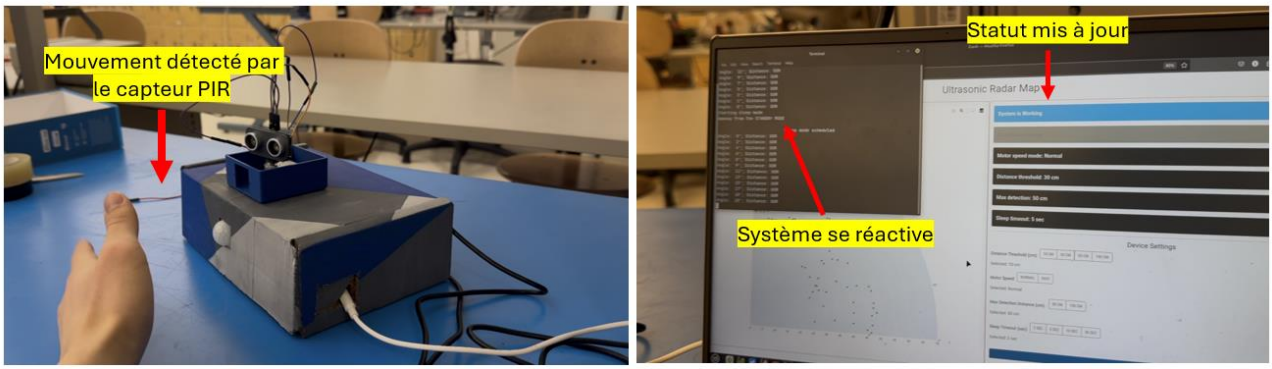

Figure 11 : *Réactivation du système après le mode veille lorsqu'il a du mouvement détecté par le capteur PIR*

Les objets détectés apparaissent sous forme de points sur une visualisation radar indiquant leur angle et leur distance de détection *(voir Figure 12)*. À droite sur l'application, le statut du système ainsi que les paramètres actuels sont affichés. Étant donné que la détection actuelle (32 cm) est plus grande que le seuil de déclenchement de l'alarme (30 cm), le message *"No nearby objects"* est affiché et l'alarme n'est pas déclenchée.

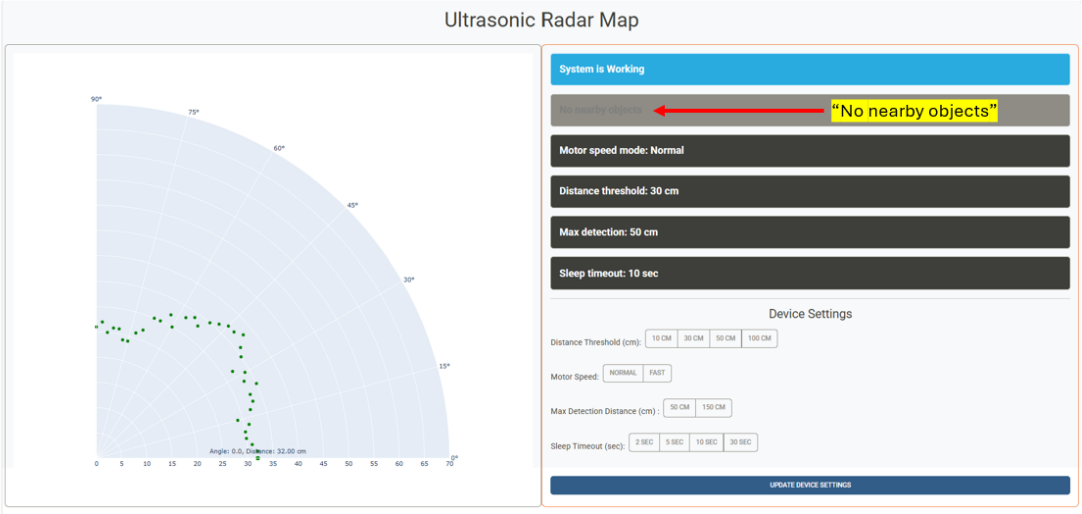

Figure 12 : *Capture d'écran de l'application web en fonctionnement normal*

Les paramètres du système peuvent être modifiés via les différentes options disponibles sur l'interface web, puis sont appliqués en cliquant sur le bouton *"Update device settings"*. Les changements prennent effet presque immédiatement dans le système et sont également mises à jour sur l'application. Dans l'exemple ci-dessous, des nouveaux paramètres sont sélectionnés et mises à jour dans le système à travers la connexion Bluetooth. Maintenant, pour une détection d'un objet à une distance de 37 cm, un message d'alerte s'affiche car le seuil de distance minimale de l'alarme a été ajusté à 50 cm *(voir Figure 13)*.

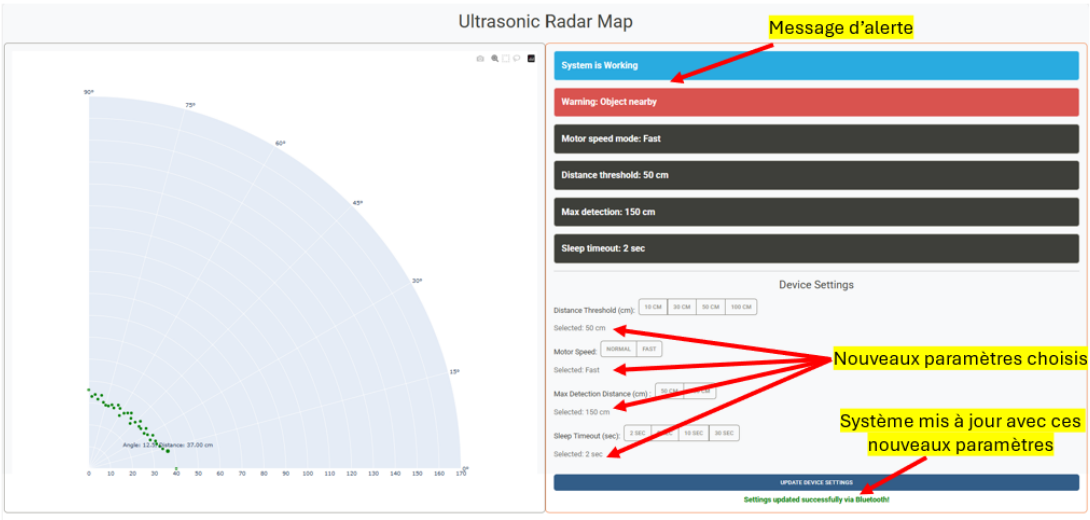

Figure 13 : *Capture d'écran de l'application web après une mise à jour des paramètres*

ANALYSE

Le système développé permet de détecter des objets à proximité et de transmettre les informations à une application web en temps réel. Il intègre des fonctionnalités intelligentes qui le rendent adapté à divers scénarios, tels que le déclenchement d'une alarme lorsque des objets sont trop proches ou l'activation d'un mode veille en l'absence de détection. Ce système embarqué a été spécialement conçu pour des applications de surveillance et de sécurité. Il intègre également des concepts de l'Internet des objets (IoT), lui permettant de communiquer avec d'autres dispositifs connectés.

Validation

Toutes les exigences initialement définies ont été satisfaites, bien que plusieurs améliorations puissent être apportées pour rendre le système plus robuste et performant. Le Tableau 2 présente ces exigences ainsi que les résultats de validation.

Tableau 2 : *Liste des résultats de validation pour les exigences.*

Exigences	Procédure de Validation	Résultat
Détection des objets	Placer des objets autour du système et observer si le capteur les détecte correctement.	Le capteur détecte de manière fiable les objets dans la proximité.
Alarme de proximité	Placer des objets à une distance inférieure ou égale à 30 cm (zone intrusive) et vérifier l'activation du buzzer.	Le buzzer s'active presque instantanément, et un message d'alerte apparaît sur l'application web indiquant la présence d'un objet proche.
Détection en temps réel	Placer des objets autour du système et vérifier la détection en temps réel.	Les objets sont détectés en temps réel, avec un délai de traitement imperceptible à l'œil nu.
Balayage continu	Laisser le système fonctionner pendant au moins 30 minutes et vérifier la continuité du balayage et de la détection d'objets.	Le système maintient un fonctionnement stable, continue à scanner et à détecter les objets sans interruption.
Transmission des données	Vérifier que les données sont correctement transmises à l'application web via la communication sans fil.	Les données sont envoyées très rapidement et de manière fiable à l'application web par transmission Bluetooth à travers l'ESP32.
Visualisation des détections	Observer les détections effectuées par le capteur et s'assurer qu'elles sont affichées sur l'application web.	Les informations de détection sont correctement affichées sur l'application web en temps réel.
Gestion de l'énergie	Bloquer les détections du capteur ultrasonique pendant 10 secondes pour simuler l'absence d'objets, puis vérifier l'entrée en mode veille. Tester la réactivation du système en simulant un mouvement devant le capteur PIR.	Après 10 secondes sans détection, le système passe en mode veille et se réactive instantanément à la détection d'un mouvement par le capteur PIR.

Configuration du système à distance	Configurer différents paramètres du système sur l'application web et vérifier que le système change selon ces configurations.	Après avoir soumis la sélection des paramètres, un message de confirmation apparaît pour indiquer que le système a bien reçu les nouvelles configurations. Le comportement du système est ensuite vérifié afin de s'assurer qu'il s'adapte correctement aux paramètres définis.
Fiabilité	Comparer les distances mesurées par le système avec les distances réelles des objets placés autour du capteur.	Les mesures du capteur ultrasonique sont précises jusqu'à la centième de centimètre.
Portabilité	Vérifier que le système peut être transporté avec un risque minimal de déconnexion des câbles et des modules matériels.	Le support et le boîtier 3D offrent une portabilité satisfaisante, protégeant les composants tout en maintenant les connexions intactes. Cependant, les câbles peuvent se déconnecter en cas de mouvements brusques. De plus, les emplacements de déploiement du système sont limités par la nécessité d'une connexion à une alimentation murale.
Latence faible	Comparer le délai entre la détection des objets et l'affichage des données sur l'application web.	Le délai est extrêmement faible, rendant la latence presque imperceptible à l'utilisateur.

Analyse Temporelle

Concernant l'analyse temporelle, le système suit un ordonnancement déterministe garantissant des temps de réponse adaptés aux contraintes d'un système embarqué en temps réel. Toutefois, il s'agit plutôt d'un système à temps réel mou, car un léger retard dans l'affichage d'une détection est tolérable. En effet, les détections se produisent très rapidement, bien plus vite que l'utilisateur ne pourrait les observer. Le système est également conçu pour être réactif, afin de détecter un objet et de transmettre immédiatement les informations pertinentes (distance et angle) avec la connexion série avant la prochaine détection. Cette réactivité est assurée par le logicielle qui exécute les instructions continuellement dans l'ordre strict suivant :

1) Obtenir la mesure de distance du capteur ultrasonique.
2) Obtenir l'angle du moteur.
3) Transmettre les données en série à l'ESP32.

Cette séquence d'instructions garantit que les informations sont envoyées au module suivant avant la réception de nouvelles données de détection. Ainsi, le système réagit à chaque détection avant que la suivante ne soit traitée. Il est important de noter que cette réactivité ne garantit pas que les données seront affichées sur l'application web avant la prochaine détection, mais elle assure qu'elles seront transmises au ESP32, chargé de la communication Bluetooth, avant la détection suivante. L'ordonnancement des tâches repose sur le noyau FreeRTOS, où la tâche de mesure du capteur ultrasonique est exécutée périodiquement avec une priorité élevée, garantissant des mesures rapides. La gestion du moteur est cadencée de manière à éviter toute latence excessive,

assurant un balayage fluide et synchronisé avec la collecte des données. De plus, la transmission des données par UART et Bluetooth est planifiée de manière non bloquante pour ne pas perturber l'exécution des tâches de détection. Les interruptions sont utilisées pour signaler qu'un nouveau paramètre sélectionné par l'utilisateur doit être appliqué au système, garantissant ainsi une réaction plus rapide pour modifier ce dernier.

Analyse du fonctionnement

Pour faire l'analyse approfondie du système, les critères de sureté de fonctionnement ont été évalué. Le système tolère certaines fautes et pannes, telles que la détection de données inattendues ou la prévention des configurations erronées et non permises par l'utilisateur sur l'application web. En cas de perte de connexion entre les composants Bluetooth, les modules tentent automatiquement de se reconnecter continuellement. Un petit système de surveillance a de nombreuses applications pratiques dans divers domaines. Il peut être utilisé pour la sécurité domestique, permettant de surveiller les entrées et les zones sensibles d'une maison ou d'un appartement. Dans les magasins, il aide à prévenir le vol et à assurer la sécurité en surveillant les zones à risque. Grâce à sa conception compacte, ce système permet aux étudiants de surveiller leurs effets personnels pendant leurs sessions d'étude ou lorsqu'ils s'absentent momentanément, par exemple pour réchauffer leur repas, leur offrant ainsi une meilleure sécurité et offre une protection contre le vol.

1) Fiabilité

Les données pour les détections sont justes et fournissent des mesures assez précises de la distance et de l'angle des objets détectés. Cependant, la fiabilité du système peut être affectée par des facteurs externes tels que les interférences sonores, les obstacles absorbant les ultrasons ou un mauvais alignement du capteur ultrasonique.

2) Disponibilité

Le système est conçu pour rester opérationnel tant qu'il détecte des objets à proximité. Lorsqu'il n'y a pas de détection pendant 10 secondes, il entre en mode veille pour économiser de l'énergie, mais il est immédiatement réactivé par le capteur PIR en cas de mouvement détecté. Cette approche garantit une disponibilité optimale tout en minimisant la consommation d'énergie. Toutefois, en cas de panne matérielle (défaillance du capteur ultrasonique, du moteur pas-à-pas ou du microcontrôleur), la disponibilité du système peut être compromise. Des tests périodiques et des diagnostics en temps réel pourraient être mis en œuvre pour détecter ces problèmes rapidement. Une redondance dans le matériel, notamment les capteurs, améliorerait le système, le rendant plus tolérant aux pannes. Une dissimilarité entre les composants redondants pourrait aussi être utilisée pour réduire la probabilité d'une défaillance du système.

3) Maintenabilité

L'architecture modulaire du système facilite la maintenance. Chaque composant (capteur ultrasonique, moteur pas-à-pas, buzzer, capteur PIR et application web) est indépendant, ce qui permet d'identifier facilement les pannes et de les corriger sans impacter l'ensemble du système.

Le code est documenté de manière claire avec des commentaires pour simplifier les futures modifications ou mises à jour. Des informations sont aussi incluses sur le répertoire GitHub dans la section *Wiki.*

4) Sécurité

Il est difficile de prévoir et développé une solution à 100 pourcent tolérant aux pannes, mais des mécanismes ont été mise en place pour réduire la chance d'avoir des erreurs qui pourrait provoquer des fautes et puis des pannes dans le système. Pour la mesure des distances avec le capteur ultrasonique, les distances sont vérifiées avant d'être envoyés vers l'application web. Il faut que la distance mesurée et obtenue soit dans la plage attendue (dans l'intervalle de la valeur minimum et maximum) avant d'être envoyé au ESP32. Si ce n'est pas le cas, la valeur est mise à -1 pour indiquer une mesure erronée ou un objet hors de la distance attendue (out-of-range). De manière similaire, la mesure de l'angle est restreinte dans la plage appropriée pour éviter les valeurs incohérentes. De plus, les configurations possibles du système proposées à l'utilisateur sur l'application web sont limitées à certaines options pour la prévention d'entrée de valeurs incorrectes ou erronées.

5) Testabilité

Le système a été testé dans diverses conditions pour valider la précision des mesures et la réactivité de l'alerte sonore. Différentes fonctionnalités ont été ajoutées au système pour faciliter le processus de test et de débogage. Des LED ont été installées pour s'allumer lorsqu'un mouvement est détecté par le capteur PIR, ainsi qu'une LED supplémentaire qui s'active lorsqu'un objet est détecté par le capteur ultrasonique à une distance trop proche (≤ seuil de distance sélectionné). Ces LED ont permis une meilleure visualisation du fonctionnement du système, notamment lors de la phase de validation et de développement, afin de s'assurer du bon fonctionnement des sous-modules individuels. De plus, la connexion UART a été utilisée pour transmettre les données en série, permettant ainsi une visualisation en temps réel des distances et des angles mesurés, qui sont ensuite envoyés pour la transmission Bluetooth *(voir Figure 14)*.

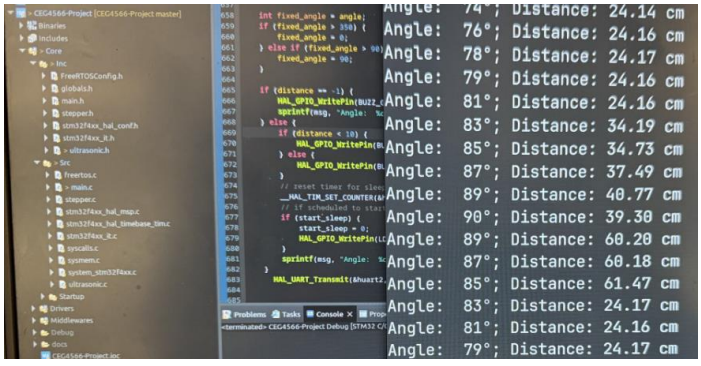

Figure 14 : *Fenêtre du terminal utilisée pour le débogage*

Les différentes exigences fonctionnelles et non-fonctionnelles ont été validées *(voir Tableau 2)*. Dans le futur, pour améliorer davantage la testabilité, des tests unitaires et d'intégration automatisés pourraient être développés et exécutés de manière régulière. Ces tests réguliers permettraient de

détecter rapidement toute défaillance ou anomalie dans le fonctionnement du système. En cas de défaillance, des notifications automatiques pourraient être envoyées pour faciliter la correction rapide des erreurs et maintenir un haut niveau de fiabilité opérationnelle.

Améliorations

Le produit final du projet est performant, mais plusieurs améliorations pourraient être implémentées pour optimiser davantage sa fiabilité et performance. Cette section explore les aspects du système qui peuvent être amélioré et des solutions pour améliorer ceux-ci.

Portée de communication limitée

Dans la configuration existante, une connexion Bluetooth est utilisée pour la communication entre le système embarqué et l'application web, ce qui limite la portée du système. Cependant, avec une connexion Wi-Fi, le système pourrait communiquer à des distances plus longues et d'envoyer et de recevoir des données à des vitesses plus élevées, permettant une interaction en temps réel plus fluide. Cette amélioration augmentera la distance qu'un utilisateur peut surveiller son environnement.

Intégration avec des applications IoT

L'ajout de capacités additionnels IoT permettraient au système de stocker et d'analyser les données dans le cloud, tout en offrant une intégration avec des frameworks existants tels que Home Assistant ou Apple HomeKit. Cette intégration faciliterait la surveillance à long terme avec une historique des détections et la gestion avancée des alertes. De plus, cela ouvrirait la possibilité d'envoyer des notifications intelligentes et d'intégrer le système dans des applications de sécurité existante.

Meilleur Portabilité

Le système dépend d'une source d'alimentation qui vient directement du mur ou d'un ordinateur portable. Ceci limite la portabilité du système qui est une exigence importante. Bien que le système soit quand même portable, il sera meilleur de remplacer l'alimentation actuelle par une solution plus mobile tel qu'une batterie rechargeable. Cela améliorait la mobilité du système et élargirait ses applications possibles. De plus, une application mobile pourrait également être conçue pour offrir une meilleure accessibilité à l'utilisateur.

Étendre les capacités de l'application web

Dans le système existant, l'utilisateur peut visualiser les détections en temps réel, ainsi que les alertes en cas d'objet qui sont proches. L'application permet aussi à l'utilisateur d'ajuster les paramètres du système à distance. Cependant, il serait pertinent d'étendre les capacités de l'application web en ajoutant plus de fonctionnalités comme l'enregistrement des données pour permettre un suivi historique. Par ailleurs, l'intégration de l'application avec des bibliothèques existantes, comme mentionné précédemment, pourrait également apporter des avantages supplémentaires.

CONCLUSION

En conclusion, un système de surveillance capable de détecter les objets dans l'environnement continuellement et de déclencher une alarme en cas d'intrusion a été conçu. Le système offre plusieurs aspects intelligents tel que la communication sans fil, l'affichage des données sur une application web, la configuration du système à distance et la conservation de l'énergie lorsqu'il n'a aucun objet dans les alentours. Le système peut servir différents buts et s'adapte à diverses situations d'utilisation. Le système offre une bonne plateforme et une solution sur laquelle il est possible de bâtir et d'améliorer, et dans le futur plusieurs améliorations pourront être ajoutées pour rendre le système plus robuste et plus fiable.

RÉFÉRENCES

[1] The Open Group, «The ArchiMate® Enterprise Architecture Modeling Language,» [En ligne]. Available: https://www.opengroup.org/archimate-forum/archimate-overview.

[2] The Data Visualisation Catalogue, «Radar Chart [Image pris de cette référence],» [En ligne]. Available: https://datavizcatalogue.com/methods/radar_chart.html.

[3] Espressif Systems, «ESP32-DevKitC-32E [Image pris de cette référence],» Mouser Electronics, [En ligne]. Available: https://www.mouser.ca/ProductDetail/Espressif-Systems/ESP32-DevKitC-32E?qs=GedFDFLaBXFpgD0kAZWDrQ%3D%3D.

[4] Isaac, «Buzzer: everything about this device to emit sound [Image pris de cette référence],» HardwareLibre, [En ligne]. Available: https://en.hwlibre.com/buzzer/.

[5] STMicroelectronics, «STMicroelectronics STM32 Nucleo Development Boards [Image pris de cette référence],» Mouser Electronics, [En ligne]. Available: https://www.mouser.ca/new/stmicroelectronics/stm-nucleo-development-boards/.

[6] S. R. H. S. Sugamya Katta, «Passive Infrared Sensor [Image pris de cette référence],» Science Direct, 2022. [En ligne]. Available: https://www.sciencedirect.com/topics/computer-science/passive-infrared-sensor.

[7] ThePiHut, «Small Reduction Stepper Motor - 5VDC 32-Step 1/16 Gearing [Image pris de cette référence],» [En ligne]. Available: https://thepihut.com/products/small-reduction-stepper-motor-5vdc-32-step-1-16-gearing.

[8] Circuito Team, «HC-SR04 Ultrasonic Sensor [Image pris de cette référence],» circuito.io blog, 26 août 2018. [En ligne]. Available: https://www.circuito.io/component/hc-sr04/.

[9] D. Castillo, «Develop Data Visualization Interfaces in Python With Dash,» Real Python, 2 février 2025. [En ligne]. Available: https://realpython.com/python-dash/.

[10] H. Blidh, «bleak,» GitHub, 7 mai 2024. [En ligne]. Available: https://github.com/hbldh/bleak.

ANNEXE

Répertoire GitHub : code principale du STM32 et code application web + et du ESP32

```c
float ultrasonic_get_distance() {
    // Trigger the ultrasonic pulse
    __HAL_TIM_ENABLE(ultrasonic_conf.timer);

    // Wait for the OS flag to indicate that the echo measurement is done
    osThreadFlagsWait(ultrasonic_conf.flag_os_thread, osFlagsWaitAny, osWaitForever);

    // Reset the OS flag after the measurement is done
    osThreadFlagsClear(ultrasonic_conf.flag_os_thread);

    float distance;

    // Calculate the distance by checking the time difference between the start and end times of the echo
    if (*ultrasonic_conf.echo_start_time < *ultrasonic_conf.echo_end_time) {
        // Normal case: echo_end_time is after echo_start_time
        distance = (*ultrasonic_conf.echo_end_time - *ultrasonic_conf.echo_start_time) * 0.034 / 2;
    } else {
        // Edge case: echo_end_time might have wrapped around
        distance = ((0xFFFFFF - *ultrasonic_conf.echo_start_time) - *ultrasonic_conf.echo_end_time) * 0.034 / 2;
    }

    // Check if the distance is within the valid range
    if (distance > U_MAX_DISTANCE || distance < U_MIN_DISTANCE) return -1;

    return distance;
}
```

Figure 15 : *Fonction du librairie ultrasonic pour obtenir la distance de la détection*

```c
void stepper_half_drive()
{
    HAL_GPIO_WritePin(S_IN1_PORT, S_IN1_PIN, half_drive_sequence[cur_step][0]);
    HAL_GPIO_WritePin(S_IN2_PORT, S_IN2_PIN, half_drive_sequence[cur_step][1]);
    HAL_GPIO_WritePin(S_IN3_PORT, S_IN3_PIN, half_drive_sequence[cur_step][2]);
    HAL_GPIO_WritePin(S_IN4_PORT, S_IN4_PIN, half_drive_sequence[cur_step][3]);

    if (stepper_conf.direction == 0)
        cur_step = (cur_step + 1) % 8;
    else
        cur_step = (cur_step == 0) ? 7 : cur_step - 1;
}


void stepper_full_drive(void)
{
    // Drive the stepper motor pins using the current full-drive sequence
    HAL_GPIO_WritePin(S_IN1_PORT, S_IN1_PIN, full_drive_sequence[cur_step][0]);
    HAL_GPIO_WritePin(S_IN2_PORT, S_IN2_PIN, full_drive_sequence[cur_step][1]);
    HAL_GPIO_WritePin(S_IN3_PORT, S_IN3_PIN, full_drive_sequence[cur_step][2]);
    HAL_GPIO_WritePin(S_IN4_PORT, S_IN4_PIN, full_drive_sequence[cur_step][3]);

    // Update current step and angle based on the desired direction.
    if (stepper_conf.direction == 0) {
        // Clockwise rotation: increment step and add angle
        cur_step = (cur_step + 1) % 4;
        stepper_conf.angle += 0.17578125;
    } else {
        // Counter-clockwise rotation: decrement step and subtract angle
        cur_step = (cur_step == 0) ? 3 : cur_step - 1;
        stepper_conf.angle -= 0.17578125;
    }

    // Normalize the angle so that it stays within 0 to 360 degrees
    stepper_conf.angle = fmod(stepper_conf.angle, 360.0);
    if (stepper_conf.angle < 0)
        stepper_conf.angle += 360.0;
}
```

Figure 16 : *Capture d'écran du code de la librairie créée pour le contrôle du moteur (1)*

```c
void stepper_change_direction() {
    if (stepper_conf.direction == 0) {
        stepper_conf.direction = 1;
        cur_step = (cur_step == 0) ? (sizeof(half_drive_sequence) / sizeof(half_drive_sequence[0])) - 1 : cur_step - 1;
    } else {
        stepper_conf.direction = 0;
        cur_step = (cur_step + 1) % (sizeof(half_drive_sequence) / sizeof(half_drive_sequence[0]));
    }
}


void stepper_set_speed(uint16_t counter) {
    __HAL_TIM_SET_AUTORELOAD(stepper_conf.timer, counter-1);
    stepper_conf.timer->Instance->EGR |= TIM_EGR_UG;
}


float stepper_get_angle() {
    return stepper_conf.angle;
}
```

Figure 17 : *Capture d'écran du code de la librairie créée pour le contrôle du moteur (2)*

```c
void UltrasonicTask(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
  float distance = ultrasonic_get_distance();

  // Get the current angle (in degrees) and round it to the nearest integer.
  int angle = (int)(stepper_get_angle() + 0.5f);

  int fixed_angle = angle;
  if (fixed_angle > 350) {
    fixed_angle = 0;
  } else if (fixed_angle > 90) {
    fixed_angle = 90;
  }

  if (distance == -1) {
    HAL_GPIO_WritePin(BUZZ_GPIO_Port, BUZZ_Pin, GPIO_PIN_RESET);
    sprintf(msg, "Angle:  %d°; Distance: OOR\r\n", fixed_angle);
  } else {
    if (distance < 30) {
      HAL_GPIO_WritePin(BUZZ_GPIO_Port, BUZZ_Pin, GPIO_PIN_SET);
    } else {
      HAL_GPIO_WritePin(BUZZ_GPIO_Port, BUZZ_Pin, GPIO_PIN_RESET);
    }
    // reset timer for sleep mode
    __HAL_TIM_SET_COUNTER(&htim10, 0);
    // if scheduled to start sleep and object detected, unschedule sleep
    if (start_sleep) {
      start_sleep = 0;
      HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    }
    sprintf(msg, "Angle:  %d°; Distance: %0.2f cm\r\n", fixed_angle, distance);
  }
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
```

Figure 18 : *Tâche principale du programme main.c sur STM32 (1)*

```
    // Create a buffer to hold binary data
    uint8_t data[8];
    memcpy(data, &fixed_angle, sizeof(fixed_angle)); // Copy int (4 bytes)
    memcpy(data + sizeof(fixed_angle), &distance, sizeof(distance)); // Copy float (4 bytes)

    // Transmit binary data
    HAL_UART_Transmit(&huart6, data, sizeof(data), HAL_MAX_DELAY);
    if ((angle_start && angle > 350) || (!angle_start && angle >= 90)) {
      // if at the start position and sleep is scheduled, go into standby mode
      if (angle_start) {
        if (start_sleep) {
          sprintf(msg, "Starting sleep mode\r\n");
          HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
          __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);
          HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN1);
          HAL_PWR_EnterSTANDBYMode();
        }
        angle_start = 0;
      } else angle_start = 1;
      stepper_disable();
      osDelay(1000);
      stepper_change_direction();
      stepper_enable();
    }
  osDelay(100);
  }
  /* USER CODE END 5 */
}
```

Figure 19 : *Tâche principale du programme main.c sur STM32 (2)*

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    memcpy(&speed_mode, &RxData[0], 4);
    memcpy(&threshold, &RxData[4], 4);
    memcpy(&max_range, &RxData[8], 4);
    memcpy(&duration, &RxData[12], 4);

    // speed mode
    if (speed_mode == 1) {
      stepper_set_speed(6500);
    } else {
      speed_mode = 0;
    stepper_set_speed(11000);
    }

    // threshold
    if (threshold == 10) {

    } else if (threshold == 50) {

    } else if (threshold == 100) {

    } else {
      threshold = 30;
    }

    // max range
    if (max_range == 150) {

    } else {
      max_range = 50;
    }
```

Figure 20 : *Fonction pour la mise à jour des paramètres configurés sur l'application web (1)*

```
    // duration
    if (duration == 2) {
      __HAL_TIM_SET_AUTORELOAD(&htim10, 4000-1);
      htim10.Instance->EGR |= TIM_EGR_UG;
    } else if (duration == 5) {
      __HAL_TIM_SET_AUTORELOAD(&htim10, 10000-1);
      htim10.Instance->EGR |= TIM_EGR_UG;
    } else if (duration == 30) {
      __HAL_TIM_SET_AUTORELOAD(&htim10, 60000-1);
      htim10.Instance->EGR |= TIM_EGR_UG;
    } else {
      duration = 10;
      __HAL_TIM_SET_AUTORELOAD(&htim10, 20000-1);
      htim10.Instance->EGR |= TIM_EGR_UG;
    }


  sprintf(msg, "Updated settings: %d,%d,%d,%d\r\n", speed_mode, threshold, max_range, duration);
  HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);

    HAL_UART_Receive_IT(&huart6, RxData, 16);
}
```

Figure 21 : *Fonction pour la mise à jour des paramètres configurés sur l'application web (2)*

```
void setup() {
    Serial.begin(115200);
    Serial1.begin(115200, SERIAL_8N1, 0, 1);  // UART TX=0, RX=1

    // Initialize BLE
    BLEDevice::init("ESP32_BLE");
    BLEServer *pServer = BLEDevice::createServer();
    BLEService *pService = pServer->createService(SERVICE_UUID);

    // RX Characteristic (Receive from Python)
    BLECharacteristic *rxCharacteristic = pService->createCharacteristic(
        RX_CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_WRITE
    );
    rxCharacteristic->setCallbacks(new MyCallbacks());

    // TX Characteristic (Send to Python)
    txCharacteristic = pService->createCharacteristic(
        TX_CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_NOTIFY
    );

    pService->start();
    BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
    pAdvertising->addServiceUUID(SERVICE_UUID);
    BLEDevice::startAdvertising();
    Serial.println("Waiting for BLE connection...");
}
```

Figure 22 : *Capture d'écran de la réception des données en série (UART) et la transmission des données via Bluetooth (1)*

```
void loop() {
    uint8_t data[24];

    // Read from Serial1 and send via BLE
    if (Serial1.available() >= 24) {
        Serial1.readBytes(data, 24);
        txCharacteristic->setValue(data, sizeof(data));
        txCharacteristic->notify();

        // Debug: Print received fields
        int angle;
        float distance;
        int motor_speed, alarm_threshold, max_distance, sleep_timeout;

        memcpy(&angle, data, sizeof(angle));
        memcpy(&distance, data + sizeof(angle), sizeof(distance));
        memcpy(&motor_speed, data + sizeof(angle) + sizeof(distance), sizeof(motor_speed));
        memcpy(&alarm_threshold, data + sizeof(angle) + sizeof(distance) + sizeof(motor_speed), sizeof(alarm_threshold));
        memcpy(&max_distance, data + sizeof(angle) + sizeof(distance) + sizeof(motor_speed) + sizeof(alarm_threshold), sizeof(max_distance));
        memcpy(&sleep_timeout, data + sizeof(angle) + sizeof(distance) + sizeof(motor_speed) + sizeof(alarm_threshold) + sizeof(max_distance),

        // Serial.printf("Received: Angle = %d, Distance = %.2f cm\n", angle, distance);
        // Serial.printf("Motor Speed = %d, Alarm Threshold = %d, Max Distance = %d, Sleep Timeout = %d\n", motor_speed, alarm_threshold, max_
    }

    delay(10);
}
```

Figure 23 : *Capture d'écran de la réception des données en série (UART) et la transmission des données via Bluetooth (2)*

```
class MyCallbacks : public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) override {
        std::string receivedValue = pCharacteristic->getValue();

        if (receivedValue.length() >= 16) { // Expecting at least 16 bytes
            Serial.print("Received from Python: ");

            int value1, value2, value3, value4;
            memcpy(&value1, receivedValue.data(), sizeof(value1));
            memcpy(&value2, receivedValue.data() + sizeof(value1), sizeof(value2));
            memcpy(&value3, receivedValue.data() + sizeof(value1) + sizeof(value2), sizeof(value3));
            memcpy(&value4, receivedValue.data() + sizeof(value1) + sizeof(value2) + sizeof(value3), sizeof(value4));

            Serial.printf("Value1 = %d, Value2 = %d, Value3 = %d, Value4 = %d\n", value1, value2, value3, value4);

            // Forward the raw bytes to Serial1
            Serial1.write((uint8_t*)receivedValue.data(), receivedValue.length());
        }
    }
};
```

Figure 24 : *Capture d'écran de la réception des données Bluetooth et la transmission des données en série (UART)*

```python
def update(self):
    left_idx = 0
    right_idx = 0
    if self.previous_angle < self.current_angle:
        left_idx = bisect.bisect_left(self.angles, self.previous_angle) + 1
        right_idx = bisect.bisect_right(self.angles, self.current_angle)
    elif self.previous_angle > self.current_angle:
        left_idx = bisect.bisect_left(self.angles, self.current_angle)
        right_idx = max(
            bisect.bisect_right(self.angles, self.previous_angle) - 1, 0
        )
    else:
        left_idx = bisect.bisect_left(self.angles, self.current_angle)
        right_idx = bisect.bisect_right(self.angles, self.current_angle)

    self.angles = self.angles[:left_idx] + self.angles[right_idx:]
    self.distances = self.distances[:left_idx] + self.distances[right_idx:]
    # self.timestamps = self.timestamps[:left_idx] + self.timestamps[right_idx:]

    print(
        f"Inserting self.current_angle {self.current_angle}, self.current_distance {self.current_distance}"
    )
    insert_idx = bisect.bisect_left(self.angles, self.current_angle)
    self.angles.insert(insert_idx, self.current_angle)
    self.distances.insert(insert_idx, self.current_distance)
    # self.timestamps.insert(insert_idx, self.timestamp_id)
    # self.timestamp_id += 1
```

Figure 25 : *Fonction de l'application web qui met à jour les données sur la visualisation radar*