Institut National de Radioélectricité et de Cinématographie

Enseignement technique secondaire de qualification

Accès aux études supérieures

Avenue Jupiter, 188

1190 Forest

Berceau Connecté Intelligent

Surveillance IoT d'un nourrisson par radar mmWave, capteurs optiques et vision artificielle

Réalisé pour la Crèche Lyra — Schaerbeek, Bruxelles

Projet personnel de [VOTRE NOM COMPLET]

Pour l'obtention du certificat de qualification

Tuteur : M. Ben Sellam

Année scolaire : 2025–2026

Remerciements :

Je tiens à remercier Monsieur Ben Sellam, mon tuteur, pour son accompagnement, ses conseils et sa disponibilité tout au long de ce projet.

Je remercie également mes professeurs de l'INRACI, qui m'ont transmis les bases nécessaires, aussi bien en informatique embarquée, en réseau, en système d'exploitation et en structure matérielle et logicielle.

Je remercie aussi l'équipe de la Crèche Lyra à Schaerbeek pour m'avoir inspiré ce projet et pour leur intérêt pour la sécurité des nourrissons.

Enfin, je remercie toutes les personnes qui m'ont encouragé et motivé, en particulier mes camarades de classe, avec qui j'ai pu partager mes idées et résoudre certaines difficultés.

Ce projet m'a permis de mettre en pratique l'ensemble des connaissances acquises durant mes années de formation en informatique. Je suis fier d'avoir pu réaliser un système complet de surveillance intelligent pour la sécurité des bébés.

Table des matières

I. Introduction

II. Théories

1. Matériels

1.1 Raspberry Pi 4

1.2 ESP32-C6 (XIAO Seeed Studio)

1.3 ESP32-WROVER (Freenove)

1.4 Capteur radar MR60BHA2

1.5 MAX30102 — Capteur optique BPM + SpO2

1.6 LM393 — Capteur de lumière

1.7 Buzzer passif

1.8 Caméra Pi Camera NoIR

2. Logiciels

2.1 MicroPython

2.2 Python 3 / Flask

2.3 paho-mqtt

2.4 HiveMQ (broker MQTT)

2.5 OpenCV et face_recognition

2.6 Chart.js

2.7 vsftpd (serveur FTP)

2.8 Raspberry Pi OS

2.9 Thonny IDE

2.10 Visual Studio Code

III. Topologie

IV. Explication du travail réalisé

1. Choix du projet
2. Configuration de l'ESP32-C6 avec le radar MR60BHA2
3. Configuration de l'ESP32-WROVER
4. Mise en place du broker MQTT
5. Développement du dashboard Flask (Pi 1)
6. Système de reconnaissance faciale (Pi 2)
7. Page d'enregistrement des visages
8. Services systemd et démarrage automatique

V. Conclusion

Bibliographie

Annexes

I. Introduction :

Durant ma dernière année d'étude en 6ème informatique à l'INRACI, j'ai réalisé un projet de fin d'étude (TFE) sur la surveillance intelligente des nourrissons. Ce travail a été réalisé dans un contexte scolaire, à Bruxelles, entre septembre 2025 et juin 2026. Le projet a été développé en collaboration avec la Crèche Lyra à Schaerbeek, qui accueille des nourrissons de 0 à 3 ans.

La mort subite du nourrisson (MSN) est une préoccupation majeure pour les parents et les professionnels de la petite enfance. En Belgique, on recense encore plusieurs dizaines de cas par an. Une surveillance continue du bébé pendant son sommeil, incluant sa fréquence respiratoire, son rythme cardiaque et sa saturation en oxygène, permet de détecter rapidement toute anomalie et d'alerter les personnes responsables.

Mon projet vise à concevoir et réaliser un prototype de berceau connecté intelligent capable de surveiller en temps réel les paramètres vitaux d'un nourrisson, sans aucun contact physique avec lui, grâce à un capteur radar millimétrique et des capteurs optiques. Une caméra avec reconnaissance faciale permet également d'identifier les personnes autorisées à accéder au bébé. Toutes les données sont centralisées sur un tableau de bord web accessible depuis n'importe quel appareil connecté au réseau local.

La question centrale de ce projet est la suivante : Comment mettre en place un système de surveillance non invasif, complet et en temps réel pour la sécurité d'un nourrisson, en utilisant des composants accessibles et des logiciels open source ?

Dans un premier temps, je présente le matériel et les logiciels utilisés. Ensuite, j'explique comment chaque partie du système a été programmée, installée et testée.

II. Théories :

1. Matériels :

Composant	Quantité	Utilité principale	Prix unitaire (€)	Total (€)
Raspberry Pi 4 (x2)	2	Serveur Flask (Pi 1) + Stream vidéo / reconnaissance faciale (Pi 2)	59,95	119,90
Carte SD 32 GB (x2)	2	Stockage du système d'exploitation Raspberry Pi OS	9,99	19,98
ESP32-WROVER (Freenove)	1	Interface MAX30102 + LM393 + buzzer via MicroPython	12,99	12,99
ESP32-C6 (XIAO Seeed)	1	Interface avec le radar mmWave MR60BHA2	7,99	7,99
Radar MR60BHA2	1	Mesure sans contact de la respiration et du rythme cardiaque	15,99	15,99
MAX30102	1	Mesure optique du BPM et de la saturation en oxygène (SpO2)	3,99	3,99
LM393 capteur lumière	1	Détection du niveau de luminosité (mode jour/nuit)	1,99	1,99
Buzzer passif	1	Alertes sonores locales en cas d'anomalie détectée	0,99	0,99
Caméra Pi Camera NoIR	1	Surveillance vidéo infrarouge + reconnaissance faciale	24,99	24,99
Câbles et breadboard	1	Connexions électroniques entre les composants	9,99	9,99
			TOTAL :	**218,80 €**

1.1 Raspberry Pi 4 :

Le Raspberry Pi 4 est un nano-ordinateur de la taille d'une carte de crédit. Il est équipé d'un processeur ARM Cortex-A72 (64 bits, 1,5 GHz), d'une mémoire RAM allant de 2 à 8 Go, d'une connexion Wi-Fi 802.11ac, du Bluetooth 5.0, de ports USB, d'un port Ethernet Gigabit et de broches GPIO pour connecter des composants électroniques.

Dans ce projet, j'utilise deux Raspberry Pi 4 : le premier (Pi 1, IP : 192.168.68.120) héberge le tableau de bord Flask et reçoit toutes les données des capteurs via MQTT. Le second (Pi 2, IP : 192.168.68.123) gère le flux vidéo MJPEG et la reconnaissance faciale.

Raison : Je l'ai choisi car c'est un micro-ordinateur Linux complet, capable de faire tourner Python, Flask, OpenCV et des services réseau (MQTT, FTP, HTTP) en simultané, tout en restant abordable et facile à configurer.

1.2 ESP32-C6 (XIAO Seeed Studio) :

L'ESP32-C6 est un microcontrôleur de la gamme ESP32 fabriqué par Espressif. Il dispose d'un processeur RISC-V 32 bits, du Wi-Fi 6 (802.11ax), du Bluetooth 5.0 et du Zigbee/Thread. Il possède plusieurs interfaces de communication : I2C, SPI, UART, ainsi que des broches GPIO.

Dans ce projet, l'ESP32-C6 (au format XIAO de Seeed Studio) est connecté directement au capteur radar MR60BHA2 via UART. Il reçoit les données de respiration et de fréquence cardiaque, puis les publie sur le broker MQTT public HiveMQ via Wi-Fi.

Raison : Je l'ai choisi pour sa compatibilité native avec le radar MR60BHA2, son support MicroPython et son faible coût. La connexion Wi-Fi intégrée permet une transmission sans fil directe vers le broker MQTT.

1.3 ESP32-WROVER (Freenove) :

L'ESP32-WROVER est une carte de développement basée sur le module ESP32-WROVER-E d'Espressif. Elle intègre 8 Mo de PSRAM externe, 4 Mo de Flash, une antenne Wi-Fi et Bluetooth intégrée, ainsi que de nombreuses broches GPIO disponibles. La carte Freenove inclut un régulateur de tension et une connectique facilement accessible pour les projets de prototypage.

Dans ce projet, l'ESP32-WROVER est programmé en MicroPython. Il est connecté au capteur MAX30102 via I2C (SDA=GPIO21, SCL=GPIO22), au capteur LM393 via GPIO34, et à un buzzer passif via PWM sur GPIO25. Il publie les données BPM optique, SpO2 et luminosité via MQTT.

Raison : Je l'ai choisi pour sa mémoire PSRAM étendue qui permet d'exécuter des programmes MicroPython complexes, et pour la disponibilité de ses GPIO permettant de connecter simultanément plusieurs capteurs.

1.4 Capteur radar MR60BHA2 :

Le MR60BHA2 est un capteur radar millimétrique (60 GHz) développé par Seeed Studio. Il utilise la technologie FMCW (Frequency Modulated Continuous Wave) pour détecter et mesurer sans aucun contact physique la fréquence respiratoire et le rythme cardiaque d'une personne. La communication s'effectue via une interface UART à 115200 bauds.

Sa portée de détection est de 0 à 150 cm, ce qui est parfaitement adapté à la surveillance d'un bébé dans un berceau. Le capteur fonctionne même à travers des couvertures légères, ce qui le rend totalement non invasif.

Raison : Je l'ai choisi car il permet une surveillance sans contact et sans lumière, ce qui respecte le sommeil du bébé. Il mesure directement les deux paramètres les plus critiques pour détecter une apnée : la respiration et le rythme cardiaque.

1.5 MAX30102 — Capteur optique BPM + SpO2 :

Le MAX30102 est un module capteur oxymètre et moniteur de fréquence cardiaque de Maxim Integrated. Il utilise des LEDs infrarouge et rouge pour mesurer optiquement la fréquence cardiaque (BPM) et la saturation en oxygène du sang (SpO2) en analysant les variations d'absorption de la lumière dans les tissus capillaires. La communication s'effectue via le protocole I2C.

Il constitue une deuxième méthode de mesure du rythme cardiaque, complémentaire au radar, qui offre en plus la mesure de la saturation en oxygène (SpO2). Une valeur de SpO2

inférieure à 94 % déclenche une alerte warning, et inférieure à 90 % une alerte critique avec le buzzer.

Raison : Je l'ai choisi car il ajoute la mesure de la SpO2, un paramètre essentiel pour détecter une hypoxie chez le nourrisson. Il constitue une redondance utile avec le radar pour la mesure du rythme cardiaque.

1.6 LM393 — Capteur de lumière :

Le module LM393 est un capteur de lumière digital qui utilise une photorésistance et un comparateur de tension pour détecter la présence ou l'absence de lumière. Sa sortie est binaire : 0 lorsqu'il y a de la lumière (jour) et 1 lorsqu'il fait sombre (nuit). La sensibilité peut être réglée via un potentiomètre.

Dans ce projet, le LM393 est connecté au GPIO34 de l'ESP32-WROVER. Il permet de détecter automatiquement si la pièce est en mode jour ou nuit, et de basculer le thème du tableau de bord Flask en conséquence (thème sombre la nuit, thème clair le jour).

Raison : Je l'ai choisi pour son prix très bas et sa simplicité d'utilisation. Il permet de rendre le tableau de bord adaptatif à l'environnement lumineux de la chambre du bébé, sans aucune interaction manuelle.

1.7 Buzzer passif :

Un buzzer passif est un transducteur électroacoustique qui émet un son lorsqu'on lui applique un signal PWM (Pulse Width Modulation) à une fréquence donnée. Contrairement à un buzzer actif, la fréquence du son est contrôlée par le programme, ce qui permet de jouer différentes tonalités et mélodies.

Le buzzer est connecté au GPIO25 de l'ESP32-WROVER en mode PWM. Il émet des sonneries différentes selon la gravité de l'alerte : 2 bips lents à 1500 Hz pour un avertissement (SpO2 < 94 % ou BPM anormal), et 5 bips rapides à 2500 Hz pour une alerte critique (SpO2 < 90 % ou BPM très anormal).

Raison : Je l'ai choisi pour alerter localement les puéricultrices même si elles n'ont pas accès au tableau de bord. Les différentes tonalités permettent d'identifier immédiatement la gravité de l'alerte.

1.8 Caméra Pi Camera NoIR :

La Pi Camera NoIR (No Infrared filter) est la version infrarouge de la caméra officielle Raspberry Pi. Elle est équipée d'un capteur Sony IMX219 de 8 mégapixels. L'absence de filtre infrarouge lui permet de filmer dans l'obscurité totale lorsqu'elle est couplée à une source lumineuse infrarouge (IR LED). Elle se connecte directement au port CSI du Raspberry Pi via un câble ruban.

Dans ce projet, la caméra NoIR est connectée au Pi 2. Elle capture en continu un flux vidéo MJPEG qui est retransmis via HTTP sur le port 8080. En parallèle, un algorithme de reconnaissance faciale analyse les images pour identifier les personnes connues (enfants enregistrés dans le système) et détecter les personnes non autorisées.

Raison : Je l'ai choisie pour sa compatibilité native avec le Raspberry Pi et sa capacité à filmer la nuit sans lumière visible, ce qui ne perturbe pas le sommeil du nourrisson. La librairie rpicam-vid permet un accès direct au flux vidéo.

2. Logiciels :

Composant	Quantité	Utilité principale	Prix unitaire (€)	Total (€)
MicroPython	1	Langage de programmation pour ESP32-C6 et ESP32-WROVER	Gratuit	Gratuit
Python 3 / Flask	1	Développement du tableau de bord web sur Raspberry Pi 1	Gratuit	Gratuit
paho-mqtt	1	Client MQTT Python pour recevoir les données des ESP32	Gratuit	Gratuit
HiveMQ (broker MQTT)	1	Broker MQTT public pour la transmission des données capteurs	Gratuit	Gratuit
OpenCV	1	Traitement d'image et capture du flux caméra sur Pi 2	Gratuit	Gratuit
face_recognition	1	Bibliothèque Python de reconnaissance faciale (dlib)	Gratuit	Gratuit
Chart.js	1	Bibliothèque JavaScript pour les graphiques historiques	Gratuit	Gratuit
vsftpd	1	Serveur FTP sur Pi 2 pour le dépôt des photos de visages	Gratuit	Gratuit
Raspberry Pi OS	1	Système d'exploitation Linux pour Raspberry Pi 1 et Pi 2	Gratuit	Gratuit
Thonny IDE	1	Environnement de développement MicroPython pour ESP32	Gratuit	Gratuit
Visual Studio Code	1	Éditeur de code Python pour le développement sur PC	Gratuit	Gratuit
			TOTAL :	**Gratuit**

2.1 MicroPython :

MicroPython est une implémentation légère du langage Python 3, optimisée pour fonctionner sur des microcontrôleurs avec des ressources limitées (mémoire RAM de quelques centaines de Ko). Il supporte les principales fonctionnalités de Python : fonctions, classes, modules, gestion des exceptions. Sur ESP32, il donne accès aux périphériques matériels via les modules machine (GPIO, I2C, PWM, UART) et network (Wi-Fi).

Raison : Je l'ai choisi car il permet d'écrire du code Python directement sur l'ESP32, sans avoir besoin de compiler en C. Cela accélère le développement et facilite le débogage via la console Thonny.

2.2 Python 3 / Flask :

Python est un langage de programmation interprété, orienté objet, largement utilisé pour le développement web, la data science et l'embarqué. Flask est un micro-framework web Python qui permet de créer des applications web légères avec un minimum de code. Il gère le routage HTTP, les templates et les réponses JSON.

Dans ce projet, Flask fait tourner le tableau de bord sur le Pi 1, expose les endpoints JSON (/api/data, /api/history, /api/person), sert la page principale et la page /capture pour l'enregistrement des visages.

Raison : Je l'ai choisi car c'est le langage que j'ai appris durant ma formation et Flask est simple à utiliser pour créer un serveur web avec des APIs JSON.

2.3 paho-mqtt :

paho-mqtt est la bibliothèque cliente MQTT officielle de la fondation Eclipse pour Python. Elle implémente le protocole MQTT versions 3.1 et 3.1.1 et permet de se connecter à un broker MQTT, de s'abonner à des topics et de publier des messages.

Raison : Je l'ai utilisée pour que le Raspberry Pi 1 puisse recevoir en temps réel les données publiées par les deux ESP32 sur le broker HiveMQ.

2.4 HiveMQ (broker MQTT public) :

HiveMQ est un broker MQTT disponible en version publique gratuite sur l'adresse broker.hivemq.com (port 1883). Le protocole MQTT (Message Queuing Telemetry Transport) est un protocole léger de messagerie publish/subscribe, idéal pour les objets connectés à ressources limitées. Chaque capteur publie sur un topic unique, et le Raspberry Pi s'abonne à tous les topics pour recevoir les données.

Raison : J'ai choisi HiveMQ car c'est un broker public gratuit, sans installation nécessaire, qui permet une communication MQTT entre les ESP32 et le Raspberry Pi via le réseau Wi-Fi local et Internet.

2.5 OpenCV et face_recognition :

OpenCV (Open Source Computer Vision Library) est une bibliothèque de traitement d'image et de vision par ordinateur. Elle permet de capturer le flux vidéo, de convertir les images entre différents formats de couleur (YUV420, BGR, RGB) et de traiter les frames en temps réel. La bibliothèque face_recognition (basée sur dlib) permet de détecter et d'identifier des visages dans une image en comparant des encodages faciaux.

Raison : Je les ai choisies pour mettre en place la reconnaissance faciale sur le Pi 2. Elles permettent d'identifier automatiquement les enfants enregistrés dans le système et d'alerter si une personne non autorisée est détectée.

2.6 Chart.js :

Chart.js est une bibliothèque JavaScript open source pour créer des graphiques interactifs dans un navigateur web. Elle supporte de nombreux types de graphiques (lignes, barres, camemberts...) et est responsive par défaut.

Raison : Je l'ai utilisée pour afficher l'historique des mesures (respiration, fréquence cardiaque, BPM optique, SpO2) sous forme de graphiques en courbe dans le tableau de bord, permettant de visualiser l'évolution des paramètres.

2.7 vsftpd (serveur FTP) :

vsftpd (Very Secure FTP Daemon) est un serveur FTP open source pour Linux, reconnu pour sa légèreté et sa sécurité. Il permet le transfert de fichiers entre appareils via le protocole FTP (File Transfer Protocol).

Raison : Je l'ai installé sur le Pi 2 pour permettre l'envoi de photos de visages depuis la page /capture du tableau de bord directement dans le dossier visages_connus, afin d'enregistrer de nouveaux enfants dans le système de reconnaissance.

2.8 Raspberry Pi OS :

Raspberry Pi OS (anciennement Raspbian) est le système d'exploitation officiel pour Raspberry Pi. Basé sur Debian Linux, il offre un environnement complet avec interface graphique, terminal et un gestionnaire de paquets apt. Il supporte Python 3, les bibliothèques OpenCV, face_recognition, Flask et tous les services Linux (systemd, vsftpd).

Raison : Je l'ai choisi car c'est le système d'exploitation officiel pour Raspberry Pi, bien documenté et compatible avec toutes les bibliothèques utilisées.

2.9 Thonny IDE :

Thonny est un environnement de développement Python simple et léger, particulièrement adapté aux débutants et à la programmation de microcontrôleurs. Il supporte MicroPython et permet de se connecter directement à l'ESP32 via le port série pour charger et exécuter du code, visualiser la console et déboguer en temps réel.

Raison : Je l'ai utilisé pour développer et tester le code MicroPython sur l'ESP32-C6 et l'ESP32-WROVER. Il est déjà installé sur Raspberry Pi OS et s'utilise également sur Windows.

2.10 Visual Studio Code :

Visual Studio Code est un éditeur de code gratuit développé par Microsoft. Il supporte la coloration syntaxique pour de nombreux langages (Python, JavaScript, HTML, CSS), la complétion automatique et l'extension SSH Remote qui permet de développer directement sur le Raspberry Pi depuis un PC Windows.

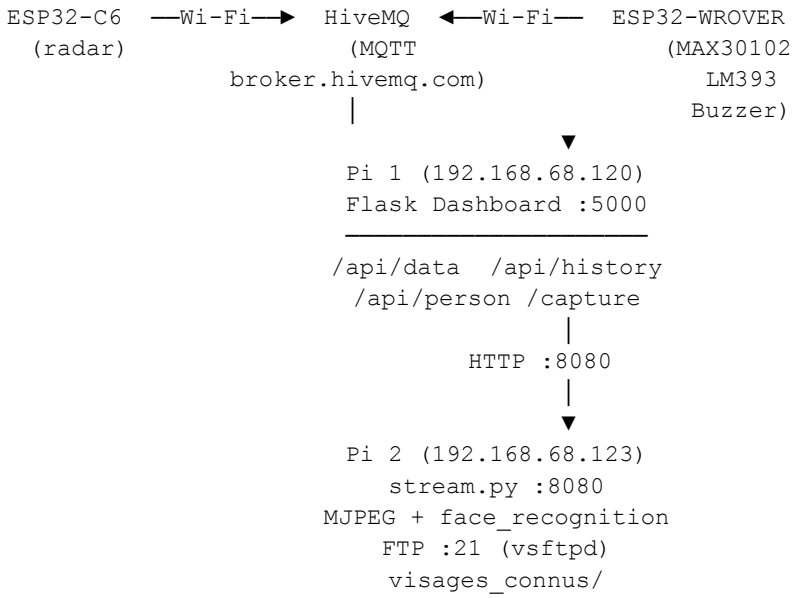
Raison : Je l'ai utilisé pour développer le code Python Flask sur le Raspberry Pi et pour éditer les fichiers de configuration Linux. L'extension SSH Remote m'a permis de travailler confortablement depuis mon PC sans quitter l'éditeur.

III. Topologie :

Le système est composé de quatre appareils connectés au même réseau Wi-Fi local (SSID : imayou). Les deux microcontrôleurs ESP32 communiquent avec le Raspberry Pi 1 via Internet en passant par le broker MQTT public HiveMQ. Le Raspberry Pi 2 communique directement avec le Raspberry Pi 1 sur le réseau local via HTTP.

Appareil	Adresse IP	Rôle	Protocole
ESP32-C6 (XIAO)	DHCP	Interface radar MR60BHA2 → publication MQTT	Wi-Fi / MQTT
ESP32-WROVER (Freenove)	DHCP	MAX30102 + LM393 + Buzzer → publication MQTT	Wi-Fi / MQTT
Raspberry Pi 1	192.168.68.120	Broker MQTT client + Tableau de bord Flask	MQTT / HTTP
Raspberry Pi 2	192.168.68.123	Stream vidéo MJPEG + Reconnaissance faciale + FTP	HTTP / FTP
Broker HiveMQ (Internet)	**broker.hivemq.com**	**Relais MQTT entre ESP32 et Pi 1**	**MQTT / TCP**

Le schéma ci-dessous illustre l'architecture complète du système :



Protocoles utilisés dans le système :

- ESP32 → broker HiveMQ : MQTT sur TCP port 1883 (Wi-Fi 2.4 GHz)
- Pi 1 → broker HiveMQ : MQTT client paho-mqtt (TCP/IP via Internet)
- Pi 2 → Pi 1 (vidéo) : HTTP multipart/x-mixed-replace (MJPEG streaming)
- Pi 2 → Pi 1 (reconnaissance) : HTTP JSON REST API
- Appareil mobile → Pi 2 : FTP port 21 (upload photo via page /capture)
- Utilisateur → Pi 1 : HTTP port 5000 (tableau de bord Flask)

Les topics MQTT utilisés suivent la convention berceau/135701/[capteur] :

- berceau/135701/cardiaque — Fréquence cardiaque radar (BPM)

- berceau/135701/respiration — Fréquence respiratoire radar (cycles/min)
- berceau/135701/distance — Distance bébé-capteur (cm)
- berceau/135701/bpm_optique — BPM optique MAX30102
- berceau/135701/spo2 — Saturation oxygène SpO2 (%)
- berceau/135701/lumiere — Niveau de luminosité LM393 (0=jour, 1=nuit)

IV. Explication du travail réalisé :

1. Choix du projet :

Pour mon année de 6ème secondaire en informatique, j'ai dû réaliser un projet de fin d'études (TFE). J'ai voulu choisir un projet qui soit concret, utile et en lien avec des sujets importants de société.

Mon idée est née d'une discussion avec l'équipe de la Crèche Lyra à Schaerbeek. Ils m'ont expliqué l'importance d'une surveillance constante des bébés pendant leur sommeil et les limites des systèmes existants. J'ai alors décidé de créer un berceau connecté intelligent qui surveille en temps réel :

- La fréquence respiratoire du bébé (sans aucun capteur en contact avec lui)
- Le rythme cardiaque par deux méthodes différentes (radar et optique)
- La saturation en oxygène du sang (SpO2)
- Le niveau de luminosité de la pièce (pour l'adaptation jour/nuit)
- La vidéo en temps réel avec reconnaissance des personnes autorisées

Ce projet m'a permis de mettre en pratique les matières vues en cours : l'informatique embarquée (ESP32, MicroPython), les réseaux (Wi-Fi, MQTT, HTTP, FTP), le développement web (Python Flask, JavaScript, Chart.js) et Linux (Raspberry Pi OS, systemd).

2. Configuration de l'ESP32-C6 avec le radar MR60BHA2 :

La première étape a été de connecter le radar MR60BHA2 à l'ESP32-C6. Le radar communique via UART à 115200 bauds. J'ai utilisé la bibliothèque MicroPython pour analyser les trames binaires envoyées par le radar, en extrayant les valeurs de respiration, de fréquence cardiaque et de distance.

Ensuite, j'ai configuré la connexion Wi-Fi sur l'ESP32-C6 et mis en place le client MQTT avec la bibliothèque umqtt.simple. L'ESP32-C6 se connecte au broker HiveMQ public (broker.hivemq.com:1883) et publie les données toutes les 2 secondes sur les topics MQTT correspondants.

J'ai également ajouté un mécanisme de reconnexion automatique : si la connexion Wi-Fi ou MQTT est perdue, l'ESP32 tente de se reconnecter automatiquement toutes les 5 secondes.

3. Configuration de l'ESP32-WROVER (MAX30102, LM393, buzzer) :

Le deuxième microcontrôleur, l'ESP32-WROVER, est responsable de trois capteurs : le MAX30102 (BPM optique + SpO2), le LM393 (luminosité) et le buzzer passif (alertes). J'ai programmé ce microcontrôleur en MicroPython avec Thonny IDE.

Le MAX30102 est connecté via I2C (SDA=GPIO21, SCL=GPIO22) et le LM393 sur GPIO34. Les données de BPM optique et de SpO2 sont publiées toutes les 5 secondes, et la valeur de luminosité toutes les 2 secondes. Le buzzer est connecté via PWM sur GPIO25.

J'ai défini des seuils d'alerte adaptés à la physiologie d'un nourrisson :

- SpO2 < 90 % → alerte critique (5 bips rapides à 2500 Hz)
- SpO2 entre 90 % et 94 % → alerte warning (2 bips lents à 1500 Hz)
- BPM < 60 ou > 180 → alerte critique
- BPM entre 60–80 ou 160–180 → alerte warning

Un problème que j'ai rencontré est que le code de publication du LM393 était placé à l'intérieur d'un bloc conditionnel qui ne s'exécutait que lorsqu'un doigt était posé sur le MAX30102. J'ai corrigé ce bug en déplaçant la publication du LM393 en dehors du bloc et en ajoutant un timer indépendant pour son intervalle de publication.

4. Mise en place du broker MQTT :

J'ai utilisé le broker MQTT public gratuit HiveMQ (broker.hivemq.com) comme intermédiaire entre les ESP32 et le Raspberry Pi 1. Les ESP32 publient leurs données sur ce broker, et le Raspberry Pi 1 s'y abonne pour recevoir les données en temps réel.

Sur le Raspberry Pi 1, j'ai utilisé la bibliothèque paho-mqtt en Python. J'ai dû adapter le code pour utiliser l'API VERSION1 de paho-mqtt 2.x, car la version plus récente requiert une déclaration explicite de la version de l'API lors de la création du client. Les données reçues sont stockées en mémoire dans un dictionnaire Python et dans un historique circulaire (deque) des 30 derniers points.

5. Développement du tableau de bord Flask (Pi 1) :

Le tableau de bord est une application web Flask qui tourne sur le Raspberry Pi 1 (port 5000). Elle est accessible depuis n'importe quel appareil connecté au réseau local en ouvrant l'adresse http://192.168.68.120:5000 dans un navigateur.

Le tableau de bord comprend les éléments suivants :

- 6 cartes de données en temps réel : respiration, BPM radar, distance, BPM optique, SpO2 et reconnaissance faciale
- Coloration automatique des cartes selon les seuils d'alerte (orange pour avertissement, rouge pour critique)
- Un flux vidéo MJPEG en direct provenant de la caméra NoIR du Pi 2
- 4 graphiques historiques en courbe (Chart.js) pour visualiser l'évolution des paramètres sur les 30 dernières mesures
- Un basculement automatique du thème jour/nuit en fonction des données du capteur LM393
- Un indicateur de statut MQTT (connecté ou déconnecté) en temps réel
- Une horloge en temps réel dans l'en-tête

Le code du tableau de bord est optimisé pour l'affichage en mode portrait (écran tourné à 90°), avec une mise en page en 2 colonnes pour les cartes et les graphiques, ce qui améliore la lisibilité sur un écran HDMI vertical.

Les données sont actualisées automatiquement toutes les 3 secondes pour les capteurs, toutes les 5 secondes pour l'historique et toutes les 2 secondes pour la reconnaissance faciale, grâce à des appels fetch() JavaScript vers les endpoints JSON de l'API Flask.

6. Système de reconnaissance faciale (Pi 2) :

Sur le Raspberry Pi 2, j'ai développé un script Python (stream.py) qui remplit deux fonctions : diffuser le flux vidéo en MJPEG et effectuer la reconnaissance faciale en temps réel.

Le premier défi a été la gestion de la caméra. La librairie rpicam-vid est utilisée pour capturer les images brutes au format YUV420, qui sont ensuite converties en BGR pour OpenCV. J'ai conçu une classe SharedCamera qui lance un seul processus rpicam-vid partagé entre tous les clients HTTP connectés. Sans cette solution, chaque nouvelle connexion au flux vidéo tentait de lancer un nouveau processus caméra, ce qui provoquait une erreur 'Pipeline handler in use by another process'.

Pour la reconnaissance faciale, j'ai utilisé la bibliothèque face_recognition (basée sur dlib). Les photos des enfants connus sont stockées dans le dossier visages_connus/ du Pi 2. À chaque frame, le système détecte les visages présents, les compare aux visages connus, et met à jour un dictionnaire de statut avec le nom de la personne reconnue, le nombre de visages détectés et un indicateur d'alerte si la personne est inconnue.

Ce statut est exposé via un endpoint JSON /api/person sur le port 8080 du Pi 2. Le Pi 1 interroge cet endpoint toutes les 2 secondes et affiche le résultat dans la carte 'Reconnaissance faciale' du tableau de bord.

Un problème que j'ai rencontré était lié au chemin du dossier visages_connus : le service systemd utilisait le mauvais nom d'utilisateur (/home/pimane au lieu de /home/younes). J'ai corrigé ce problème en modifiant le fichier de service systemd et en utilisant la commande sed pour corriger le chemin dans le script Python.

7. Page d'enregistrement des visages (/capture) :

Pour permettre aux puéricultrices d'enregistrer facilement un nouvel enfant dans le système de reconnaissance, j'ai créé une page web mobile accessible à l'adresse http://192.168.68.120:5000/capture.

Cette page permet de :

- Entrer le prénom de l'enfant dans un champ de texte
- Prendre une photo avec la caméra frontale du téléphone ou de la tablette
- Prévisualiser la photo avant de l'envoyer
- Envoyer la photo au serveur en appuyant sur 'Envoyer sur le serveur'

Lorsque l'utilisateur envoie la photo, le Pi 1 la reçoit via une requête HTTP POST multipart/form-data. Il se connecte ensuite au serveur FTP du Pi 2 (vsftpd) via la bibliothèque ftplib Python et dépose la photo dans le dossier visages_connus/ sous le nom [prénom].jpg. Le système de reconnaissance faciale prendra automatiquement en compte ce nouveau visage lors de son prochain cycle.

8. Services systemd et démarrage automatique :

Pour que le système démarre automatiquement à chaque mise sous tension du Raspberry Pi, j'ai créé des services systemd pour les deux scripts Python.

Sur le Pi 1, le service berceau.service lance automatiquement l'application Flask (app.py) au démarrage. Sur le Pi 2, le service stream.service lance le script stream.py qui gère la caméra et la reconnaissance faciale.

Un problème que j'ai rencontré était que le service systemd du Pi 2 utilisait le mauvais nom d'utilisateur dans le champ User= du fichier de service (User=pimane au lieu de User=younes). Cela empêchait le service de démarrer correctement car les fichiers et le répertoire home n'existaient pas pour l'utilisateur 'pimane' sur ce Raspberry Pi.

Les commandes Linux utilisées pour gérer les services systemd sont :

- sudo systemctl enable berceau.service — Activation du démarrage automatique
- sudo systemctl start berceau.service — Démarrage manuel du service
- sudo systemctl status berceau.service — Vérification de l'état du service
- sudo journalctl -u berceau.service -f — Affichage des logs en temps réel

V. Conclusion :

Ce projet de TFE m'a permis de réaliser un système de surveillance complet et fonctionnel pour un berceau connecté intelligent, destiné à la Crèche Lyra de Schaerbeek. L'ensemble du pipeline est opérationnel : les deux ESP32 collectent les données des capteurs et les transmettent via MQTT, le Raspberry Pi 1 les reçoit et les affiche en temps réel dans un tableau de bord web, et le Raspberry Pi 2 diffuse un flux vidéo avec reconnaissance faciale.

J'ai pu mettre en pratique les compétences acquises tout au long de ma formation : la programmation embarquée en MicroPython sur ESP32, le développement web en Python avec Flask, la vision par ordinateur avec OpenCV et face_recognition, la gestion des services Linux avec systemd, et les protocoles réseau (MQTT, HTTP, FTP).

J'ai rencontré plusieurs difficultés techniques, notamment la gestion de la caméra partagée entre plusieurs clients HTTP (problème de 'Pipeline handler in use'), les bugs de publication MQTT liés à l'API paho-mqtt 2.x, et les problèmes d'encodage de caractères lors de l'édition de fichiers via SSH. Chaque problème m'a appris à analyser les logs système, à diagnostiquer les erreurs et à trouver des solutions adaptées.

Des améliorations futures pourraient inclure : l'ajout d'une notification push vers un téléphone des parents, l'intégration d'un capteur de température et d'humidité dans le berceau, l'amélioration de l'algorithme de reconnaissance faciale pour fonctionner en conditions de faible luminosité, et la mise en conformité avec les normes de sécurité pour un usage médical.

Je suis fier d'avoir pu réaliser ce projet qui combine hardware, software et réseau dans un système réellement utile. Ce TFE représente pour moi le résultat de deux années d'apprentissage intensif à l'INRACI.

Bibliographie :

Documentation officielle MicroPython pour ESP32 : https://docs.micropython.org/en/latest/esp32/

Documentation Flask : https://flask.palletsprojects.com/

Documentation paho-mqtt Python : https://eclipse.dev/paho/files/paho.mqtt.python/html/

Seeed Studio — Documentation MR60BHA2 : https://wiki.seeedstudio.com/

Documentation face_recognition (Adam Geitgey) : https://github.com/ageitgey/face_recognition

Documentation OpenCV Python : https://docs.opencv.org/

Chart.js Documentation : https://www.chartjs.org/docs/

Raspberry Pi Documentation : https://www.raspberrypi.com/documentation/

HiveMQ Public MQTT Broker : https://www.hivemq.com/mqtt/public-mqtt-broker/

Documentation vsftpd : https://security.appspot.com/vsftpd.html

Bosch Revol — CES 2025 : https://www.bosch.com/stories/bosch-revol-smart-baby-bed/

Annexes :

Annexe A — Code MicroPython ESP32-WROVER (main.py) :

Le code ci-dessous est le programme principal de l'ESP32-WROVER. Il gère la connexion Wi-Fi, la lecture des capteurs MAX30102 (BPM + SpO2) et LM393 (lumière), la publication MQTT et le déclenchement des alertes via le buzzer.

```
from machine import Pin, I2C, PWM
import network, time
from umqtt.simple import MQTTClient

# ── Wi-Fi ──────────────────────────────────
SSID     = "imayou"
PASSWORD = "_bazz38LOI$"
# ── MQTT ───────────────────────────────────
BROKER    = "broker.hivemq.com"
CLIENT_ID = b"wrover-max30102"
TOPIC_BPM = b"berceau/135701/bpm_optique"
TOPIC_SPO = b"berceau/135701/spo2"
TOPIC_LUM = b"berceau/135701/lumiere"

# ── Composants ─────────────────────────────
buzzer = PWM(Pin(25), freq=2000, duty=0)
lm393  = Pin(34, Pin.IN)
i2c    = I2C(0, sda=Pin(21), scl=Pin(22), freq=400000)

# ── Alertes buzzer ─────────────────────────
def alerte_critique():
    for _ in range(5):
        buzzer.freq(2500); buzzer.duty(512)
        time.sleep_ms(200)
        buzzer.duty(0);    time.sleep_ms(100)

def alerte_warning():
    for _ in range(2):
        buzzer.freq(1500); buzzer.duty(512)
        time.sleep_ms(500)
        buzzer.duty(0);    time.sleep_ms(300)

# ... (lecture MAX30102 + publication MQTT)
```

Annexe B — Topics MQTT et structure des données :

Topic MQTT	Capteur	Unité	Fréquence
berceau/135701/cardiaque	Radar MR60BHA2	BPM	2 secondes
berceau/135701/respiration	Radar MR60BHA2	cycles/min	2 secondes
berceau/135701/distance	Radar MR60BHA2	cm	2 secondes
berceau/135701/bpm_optique	MAX30102 (ESP32-W)	BPM	5 secondes
berceau/135701/spo2	MAX30102 (ESP32-W)	%	5 secondes
berceau/135701/lumiere	**LM393 (ESP32-W)**	**0=jour/1=nuit**	**2 secondes**

Annexe C — Seuils d'alerte pour le nourrisson :

Paramètre	Normal	Avertissement	Critique

Respiration (radar)	30–60 cycles/min	20–30 ou 60–70	< 20 ou > 70
BPM radar	80–160 BPM	60–80 ou 160–180	< 60 ou > 180
BPM optique (MAX30102)	80–160 BPM	60–80 ou 160–180	< 60 ou > 180
SpO2 (%)	**>= 95 %**	**90–94 %**	**< 90 %**