Système de détection d'intrusion avec IoT et YOLO

Mohamed Youssef CHLENDI

December 19, 2024

Abstract

Ce projet présente un système de détection d'intrusion intégrant l'IoT avec l'algorithme de détection d'objets YOLO. Le système utilise MQTT pour la communication et propose un tableau de bord en temps réel pour la surveillance et le contrôle. Un microcontrôleur ESP32 simule l'actionnement des dispositifs, et MongoDB gère la persistance des données. Ce rapport détaille la conception, la mise en œuvre et les tests du système.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problèmatique

Assurer la sécurité dans les zones résidentielles et commerciales nécessite des systèmes automatisés capables de détecter et de réagir aux intrusions. Ce projet répond à ce besoin en utilisant des technologies IoT et de vision par ordinateur.

1.2 Objectifs

L'objectif est de concevoir un système de détection d'intrusion scalable et flexible avec les fonctionnalités suivantes :

- Détection et alertes en temps réel.
- Tableau de bord web pour la surveillance et le contrôle.
- Simulation de dispositifs avec un microcontrôleur ESP32.

Chapter 2

Conception du Système

2.1 Aperçu

Le système comprend les composants suivants :

- **Détection d'objets avec YOLO :** Détecte les objets et publie des alertes vers un broker MQTT.
- **Communication MQTT :** Facilite l'échange de messages entre les composants.
- **Tableau de bord web :** Surveille les alertes en temps réel, gère les paramètres et affiche les journaux.
- **Simulation ESP32 :** Simule des fonctionnalités d'alarme et de flash.
- **MongoDB :** Stocke les journaux de détection et les archives.

2.2 Technologies et outils

2.2.1 Technologies

- **Python**
- **Mqtt**
- **Javascript**
- **Express.js**
- **Node.js**
- **Vue.js**

2.2.2 Outils

- **GitHub :** Utilisé pour la gestion de version et le partage de code.
- **Visual Studio Code :** Environnement de développement intégré (IDE) pour écrire et débugger le code.
- **MQTT Explorer :** Outil graphique pour superviser et tester les messages publiés et reçus via MQTT.

- **MongoDB Compass :** Interface utilisateur graphique pour interagir avec la base de données MongoDB.
- **Wokwi :** Simulateur en ligne utilisé pour tester et développer des programmes ESP32 sans matériel physique.

2.3 Diagramme d'Architecture

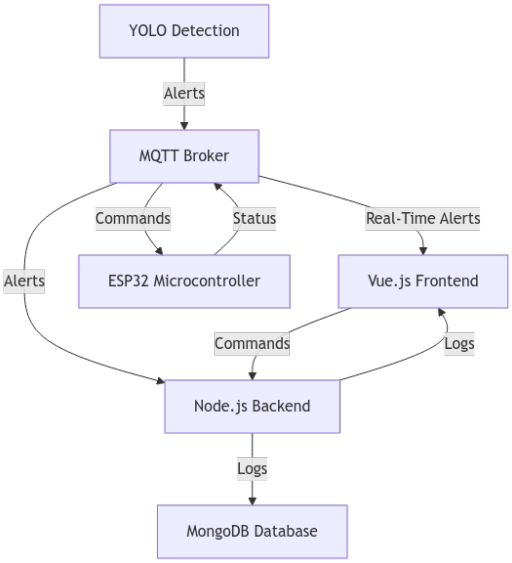

Figure 2.1: Diagramme d'architecture

Chapter 3

Mise en œuvre

3.1 Détection avec YOLO (`yolo_home_security.py`)

Ce script capture les images de la webcam, effectue la détection d'objets avec YOLOv8, et publie des alertes au broker MQTT. Le script traite en continu les images et détecte les objets avec un seuil de confiance de 50%.

Pour commencer, j'ai utilisé le modèle YOLO8N. Pour mieux répondre à l'objectif de notre système d'intrusion, nous devons affiner le modèle afin de détecter les intrusions et pas seulement les objets.

```
# Effectuer l'inf\'erence
results = model.predict(source=frame, conf=0.5, show=False)

for box in results[0].boxes:
    obj_class = int(box.cls[0])
    label = model.names[obj_class]
    confidence = box.conf[0]
    message = f"Alerte! Objet: {label}, Confiance: {confidence:.2f}"
    mqtt_client.publish(TOPIC, message)
```

Listing 3.1: Script de détection YOLO

3.1.1 Intégration avec MQTT

Le client MQTT est configuré pour publier des alertes sur le topic `home_security/alerts`, permettant une communication en temps réel avec le backend et le tableau de bord.

3.2 Logique et traitement des messages (`mqtt_subscriber.py`)

3.2.1 Fonctionnalités

Ce script s'abonne à deux topics MQTT :

- `home_security/alerts` : Pour recevoir les alertes de détection YOLO.
- `home_security/commands` : Pour recevoir les commandes du système (ex. activer la journalisation).

```
def on_message(client, userdata, msg):
    if msg.topic == COMMAND_TOPIC:
        handle_command(msg.payload.decode())
    elif msg.topic == ALERT_TOPIC:
```

```
5         save_to_mongo(msg.payload.decode())
```

Listing 3.2: Abonné MQTT

La commande handle va gérer si la commande est start nous allons activer l'enregistrement des messages dans la base de données, et si c'est stop nous allons l'arrêter.

3.2.2 Intégration avec MongoDB

Les alertes sont sauvegardées dans MongoDB avec des horodatages et des détails sur les objets détectés. Les journaux peuvent être archivés dans une collection secondaire à la demande.

3.3 Backend (`server.js`)

Le backend fournit des APIs RESTful pour les journaux, les commandes, l'état du système et la mise en cache. Voici les principales fonctionnalités :

- Récupérer les journaux depuis MongoDB.
- Publier des commandes au broker MQTT.
- Interroger l'état du système via MQTT.
- Archiver les journaux dans une collection secondaire MongoDB.

```
1  app.get("/api/logs", async (req, res) => {
2    try {
3      const logs = await DetectionLog.find().sort({ timestamp: -1 }).limit(10);
4      console.log("Logs fetched from MongoDB:", logs); // Debug line
5      res.json(logs);
6    } catch (err) {
7      console.error("Error fetching logs:", err);
8      res.status(500).send("Internal Server Error");
9    }
10 });
```

Listing 3.3: Exemple d'API Backend

3.4 Microcontrôleur ESP32 (`micropython_subscriber.py`)

Afin de simuler des tests sur un véritable microcontrôleur, j'ai utilisé le site web wokwi qui fournit des cartes et des composants permettant de simuler un véritable microcontrôleur.

Ce script simule les fonctionnalités d'alarme et de flash. L'ESP32 s'abonne au topic **home_security/commands** et exécute des actions comme l'activation de l'alarme ou le contrôle des LED pour le flash.

```
1 def on_message(topic, msg):
2     if msg == "TURN_ON_ALARM":
3         activate_alarm()
4     elif msg == "TURN_ON_FLASH":
5         activate_flash()
```

Listing 3.4: Traitement des commandes ESP32

Voici la liste détaillée des commandes/actions possibles :

- **SET_ALARM_COMMAND** : Configure le déclenchement de l'alarme en définissant la fréquence et la durée de l'alarme.
- **SET_FLASH_COMMAND** : Configure le déclenchement du flash en définissant la fréquence du flash.
- **TURN_ON_ALARM_COMMAND** : Active l'alarme.
- **TURN_ON_FLASH_COMMAND** : Active le flash.
- **TURN_OFF_ALARM_COMMAND** : Désactive l'alarme.
- **TURN_OFF_FLASH_COMMAND** : Désactive le flash.
- **GET_STATUS_COMMAND** : Envoi de l'état actuel du système au sujet home_security/status.
- **RESET_SYSTEM_COMMAND** : Réinitialise le système à son état initial.

3.5 Tableau de bord Web (`LogsDisplay.vue`)

Le tableau de bord est construit avec Vue.js et fournit :

- Une surveillance en temps réel des alertes via MQTT.
- Une gestion des journaux (consultation, rafraîchissement, mise en cache).
- Des actions pour configurer et contrôler l'alarme et les flashs.

```
methods: {
    async fetchLogs() {
        const response = await axios.get("/api/logs");
        this.logs = response.data;
    },
    async sendCommand(command) {
        await axios.post("/api/command", { command });
    }
}
```

Listing 3.5: Extrait du composant Vue.js

Chapter 4

Tests et Résultats

4.1 Scénarios de Test

- YOLO détecte des objets et publie des alertes sur MQTT.
- Les commandes activent/désactivent la journalisation et contrôlent les dispositifs ESP32.
- Le tableau de bord reflète les alertes et gère les journaux efficacement.

4.2 Captures d'écran

Dans cette section, nous présenterons différentes captures d'écran du système :

4.2.1 La détection d'un objet

Le système de détection d'objets détectera la présence d'un objet (avec un training plus poussée, il ne prendra en compte que les alertes d'intrusion) et le publiera à l'aide du mqttBroker.

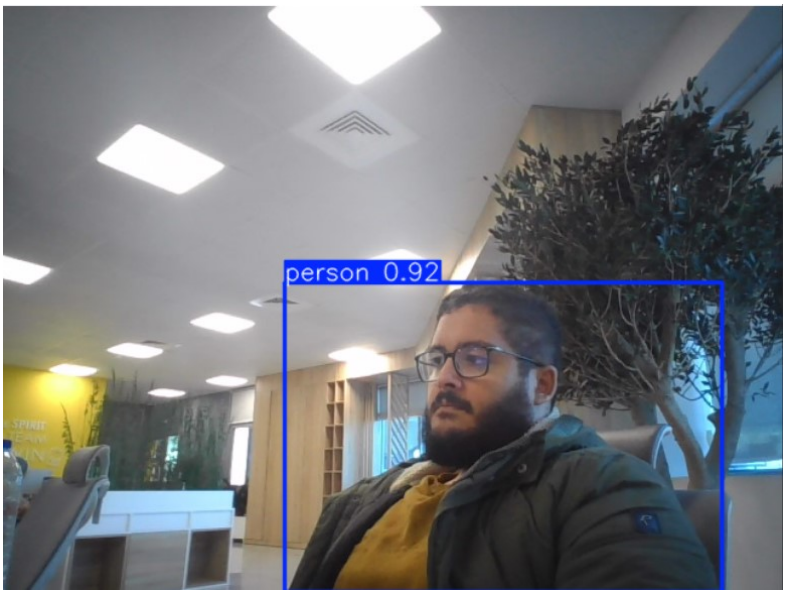

Figure 4.1: Capture de la détection d'objets

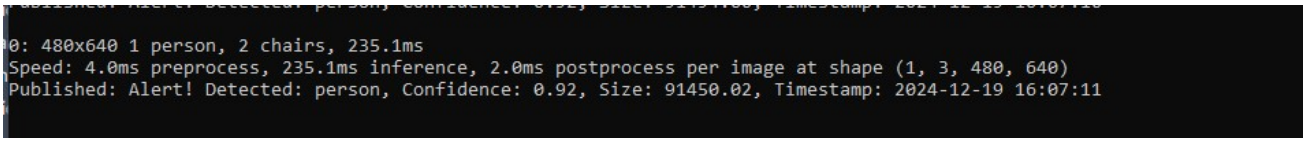

Figure 4.2: Log de la publication de detection

4.2.2 Stockage de la détection

En utilisant python, nous nous abonnons au courtier mqtt sur le thème `home_security/alerts`, sur les alertes reçues nous les stockons dans une base de données mongoDb, ici aussi l'utilisateur peut activer ou désactiver la sauvegarde des alertes à partir du tableau de bord.

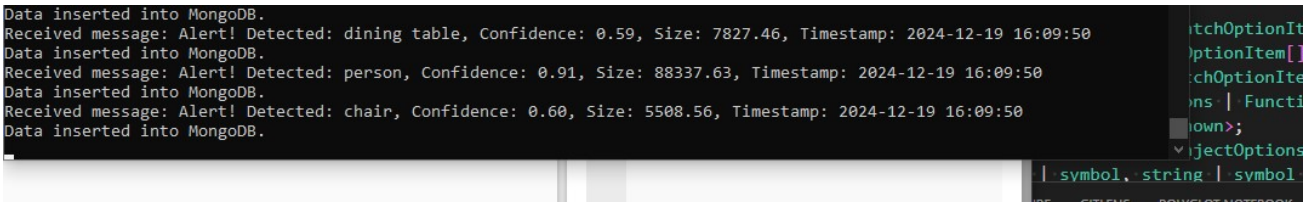

Figure 4.3: Log de l'enregistrement de l'alerte dans la base de données

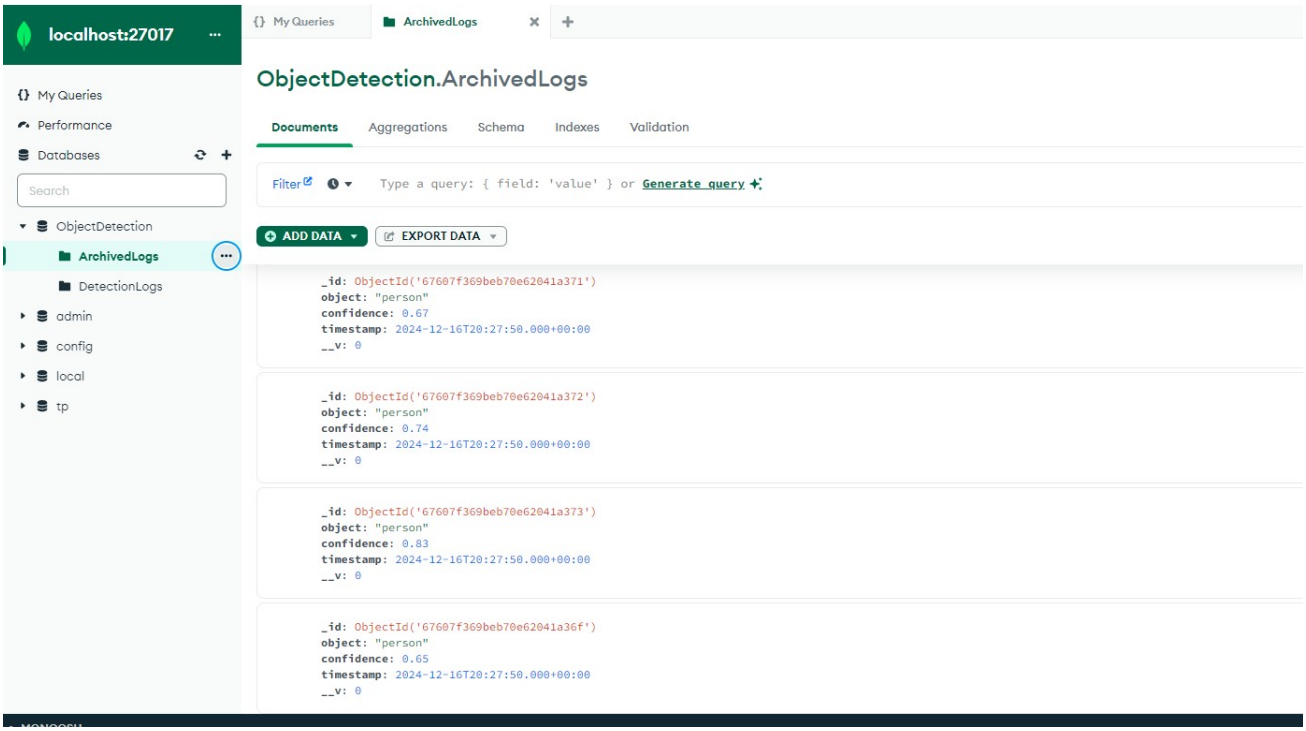

Figure 4.4: Alertes enregistrées

4.2.3 Le tableau de bord

Le tableau de bord est le lien principal avec l'utilisateur, ce tableau de bord permet à l'utilisateur de :

- 1. Visualisation de l'état du système
- 2. Affichage des alertes en temps réel

- 3. Affichage des alertes stockées
- 4. Management du système :
  - Configuration
  - Activation et désactivation des modules

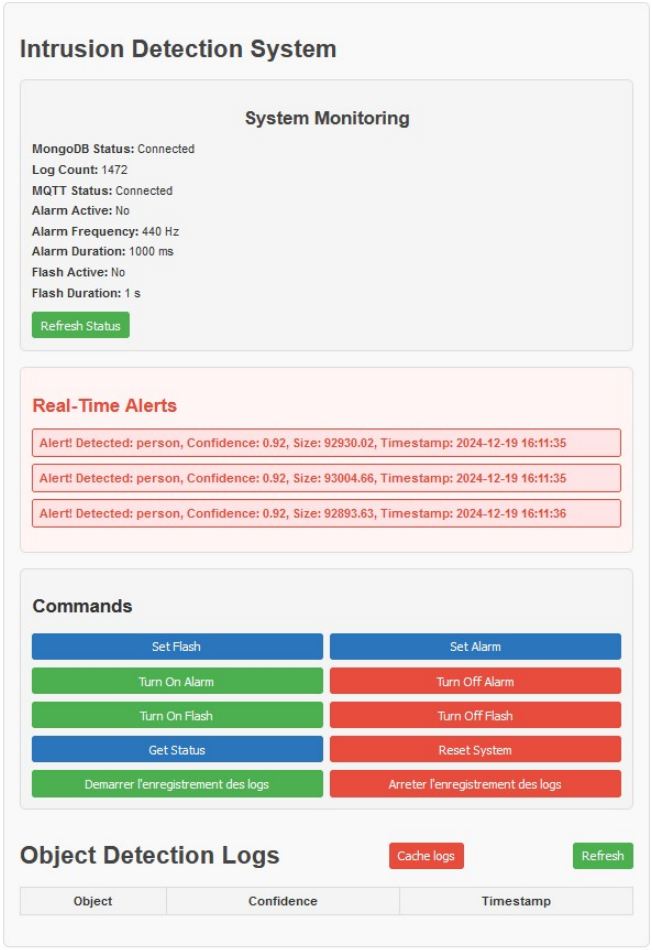

Figure 4.5: tableau de bord

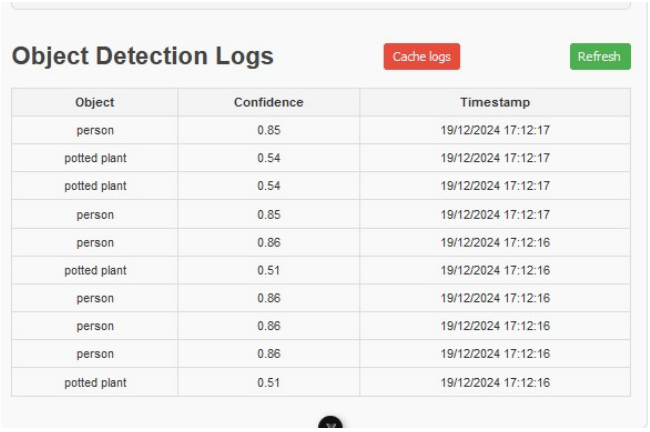

Figure 4.6: tableau de bord - Object detection logs

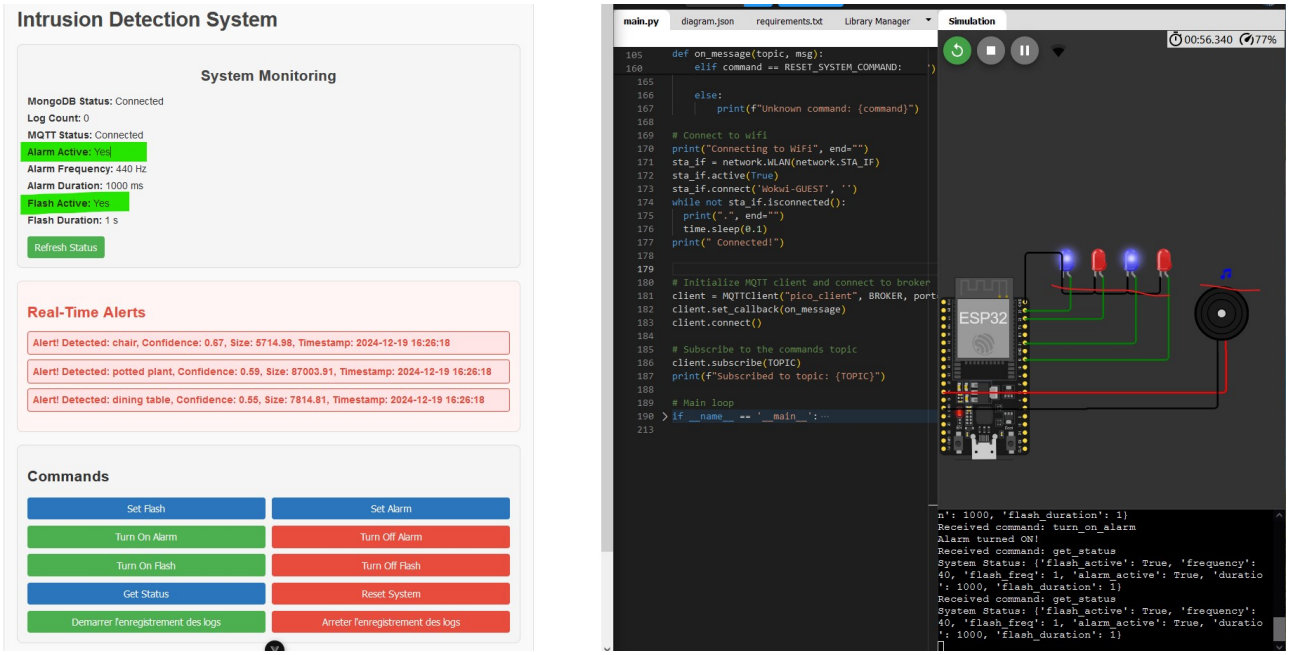

Figure 4.7: tableau de bord - Activation d'alarm et flash

Chapter 5

Conclusion et Perspectives

5.1 Réalisations

Le système intégre avec succès l'IoT, la vision par ordinateur et les technologies web pour fournir une solution robuste de détection d'intrusion.

L'ensemble du code source se trouve à l'adresse suivante : Repo

5.2 Améliorations Futures

- Intégration d'un stockage et d'une analyse des données basés sur le cloud.
- Détection avancée utilisant des modèles d'IA pour des scénarios spécifiques.
- Développement d'une application mobile pour la surveillance à distance.
- Systeme de notification a l'aide de mail/sms