Documentation Technique

Projet de Détection Automatique de Dommages et d'Équipements dans le Bâtiment

Nom du projet : Détection visuelle automatisée
Technologies utilisées : Python, Flask, Qwen-VL, OpenCV, Ngrok, iOS
Auteur : Benouadah Alaeddine

Liste des acronymes

API	Application Programming Interface
Flask	Micro-framework Python pour créer des applis web
JSON	JavaScript Object Notation (format de données)
HTML	HyperText Markup Language (langage web)
Ngrok	Tunnel pour exposer un serveur local à Internet
Qwen-VL	Modèle de vision/langage pour analyse d'image
SSH	Secure Shell (connexion distante sécurisée)
URL	Uniform Resource Locator (adresse web)
ZIP	Format compressé regroupant plusieurs fichiers

Table des matières

1 Introduction

Ce projet a pour but de faciliter l'inspection des bâtiments en automatisant la détection des défauts visibles comme les fissures, les trous ou encore les rayures. Il permet aussi de reconnaître certains équipements installés tels que les interrupteurs, prises ou extincteurs. Grâce à l'intelligence artificielle et la vision par ordinateur, les images prises sur le terrain sont analysées automatiquement. Plus besoin d'un expert pour mesurer ou identifier un dommage : l'application le fait en quelques secondes. Chaque anomalie est localisée, mesurée et annotée directement sur l'image. Ce système fonctionne avec une application iOS simple à utiliser. L'utilisateur prend une photo, l'envoie, et reçoit un rapport complet. L'objectif est de gagner du temps, d'améliorer la précision, et de réduire les coûts d'expertise. Ce projet est particulièrement utile pour les professionnels du bâtiment, les assureurs ou les inspecteurs. Il rend l'analyse visuelle technique plus rapide, plus accessible, et surtout automatisée.

2 Objectif du projet

L'objectif principal de ce projet est de rendre l'analyse visuelle des bâtiments plus rapide, automatique et accessible. Actuellement, l'inspection des dommages ou équipements nécessite une expertise humaine, du temps et parfois des outils de mesure spécifiques. Ce système vise à simplifier ce processus grâce à la technologie. Il permet de détecter automatiquement les défauts visibles sur une image (comme des fissures, trous, rayures) et de reconnaître certains équipements (prises, extincteurs, interrupteurs).

Le projet cherche à :

— Automatiser l'identification et la mesure des anomalies à partir de simples photos ;
— Fournir des informations précises comme la taille réelle, la position et le type de dommage détecté ;
— Générer des résultats clairs, compréhensibles et visuellement annotés ;
— Intégrer ce service dans une application mobile simple d'utilisation pour une exploitation sur le terrain.

En résumé, l'objectif est d'optimiser les inspections tout en réduisant le besoin d'expertise technique directe.

3 Modèle Qwen-VL

Le modèle Qwen-VL est une technologie basée sur des réseaux neuronaux avancés (transformers), entraînée sur un large corpus d'images et de textes pour reconnaître et annoter automatiquement les objets ou dommages visibles sur une image.

3.1 Fonctionnement du Modèle Qwen-VL

L'image capturée est envoyée au modèle Qwen-VL avec une requête claire (ex : « Trace une boîte autour de chaque dommage visible »). Le modèle répond en identifiant les objets et en précisant les coordonnées des boîtes englobantes.

```
model_dir = "/home/groupe/.cache/modelscope/qwen/Qwen-VL-Chat"

tokenizer = AutoTokenizer.from_pretrained(model_dir,
trust_remote_code=True)
model = AutoModelForCausalLM.from_pretrained(model_dir, device_map="cpu",
trust_remote_code=True, bf16=True).eval()
```

4 Détail du Traitement Technique (`process_depth.py`)

Le fichier `process_depth.py` constitue le coeur de l'analyse automatique des images. Il assure la décompression des données, l'interaction avec le modèle Qwen-VL, le calcul des dimensions physiques et la génération des résultats.

1. Utilisation des prompts avec Qwen-VL

Ce projet repose sur une communication entre le code Python et le modèle visuel Qwen-VL. Cette interaction est basée sur des **prompts**, c'est-à-dire des instructions textuelles précises envoyées au modèle pour obtenir une réponse.

- Exemple de prompt pour détecter et annoter les dommages :
  `{'text': 'Draw a bounding box around each visible damage in the image and label it with the name of the damage only.'}`
- Exemple de prompt pour obtenir uniquement le type de dommage :
  `{'text': 'Identify and name the type of damage visible in the image. Respond only with the damage type.'}`

  Ces instructions sont dynamiquement générées et envoyées avec l'image au modèle Qwen-VL.

Définition de Prompt

Un **prompt** est une instruction textuelle précise, formulée en langage naturel, que l'on envoie à un modèle d'intelligence artificielle pour lui indiquer exactement ce qu'on attend de lui. Dans ce projet, les prompts sont utilisés pour guider le modèle Qwen-VL dans deux tâches principales :

— La détection et l'annotation des dommages visibles dans l'image.
— L'identification du type de dommage (fissure, trou, rayure, etc.).

Ces instructions sont dynamiques, générées par le code Python, et combinées à l'image au moment de l'envoi vers le modèle.

Le modèle analyse l'image en fonction du prompt reçu et retourne soit des coordonnées (pour les boîtes englobantes), soit une réponse textuelle (comme "fissure").

2. `analyze_zip(zip_path)`

— Décompresse l'archive ZIP reçue depuis l'application iOS.
— Extrait les fichiers image (`.jpg`, `.png`) et les paramètres optiques contenus dans le fichier `calibration.json`.
— Sélectionne l'image la plus grande pour l'analyse visuelle.
— Lit les paramètres : `fx`, `fy` (longueurs focales), `cx`, `cy` (centre optique).

3. `process_image(image_path)`

— Envoie l'image au modèle Qwen-VL avec un prompt pour dessiner une boîte autour de chaque dommage visible.
— Retourne une image annotée (avec boîtes englobantes) et une description textuelle structurée.

4. `extract_bbox_dimensions(response)`

— Analyse le texte retourné par Qwen-VL pour récupérer les coordonnées de la boîte englobante (exprimées en pixels).
— Extrait les coins de la boîte : `(x1, y1)`, `(x2, y2)`.

— Convertit ces coordonnées depuis un format normalisé (valeurs sur 1000) vers les dimensions réelles de l'image (par ex. 1920x1440 pixels).

5. `process_and_resize_images(...)`

— Lit l'image de profondeur associée.
— Redimensionne l'image pour garantir une résolution standardisée.
— Calcule les profondeurs `Z1` et `Z2` aux deux coins de la boîte englobante.
— Applique les équations de projection inverse pour obtenir les coordonnées 3D physiques :

$$X = \frac{(x - cx) \cdot Z}{fx}, \quad Y = \frac{(y - cy) \cdot Z}{fy}$$

— Calcule ensuite :

$$\text{Largeur} = \sqrt{(X_2 - X_1)^2 + (Z_2 - Z_1)^2}$$

$$\text{Hauteur} = \sqrt{(Y_2 - Y_1)^2 + (Z_2 - Z_1)^2}$$

$$\text{Surface} = \text{Largeur} \times \text{Hauteur}$$

6. `get_damage_position(x1, y1, x2, y2)`

— Calcule le centre de la boîte englobante.
— Évalue sa position approximative dans l'image selon une grille 3x3 :
  — Horizontalement : à gauche / au centre / à droite
  — Verticalement : en haut / au milieu / en bas
— Retourne une description en langage naturel (ex. : "au centre à droite").

7. `process_image_text(image_path)`

— Envoie un prompt spécifique pour extraire uniquement le nom du dommage (ex : "fissure", "trou").
— Retourne une chaîne de texte interprétable directement.

Résumé du fonctionnement

Toutes ces fonctions sont orchestrées automatiquement lorsqu'un fichier ZIP est reçu. Le script traite les données, interagit avec le modèle Qwen-VL, calcule les dimensions physiques, et génère un résultat structuré, prêt à être affiché ou utilisé par l'application.

5 Calcul des dimensions physiques

Les coordonnées de la boîte englobante sont exprimées en pixels. Pour obtenir les dimensions en millimètres, une projection perspective inversée est utilisée.

Paramètres issus de `calibration.json` :

— `fx`, `fy` : longueurs focales en pixels
— `cx`, `cy` : coordonnées du centre optique

Formules utilisées :

Soit un point image (x, y) et sa profondeur Z, alors :

$$X = \frac{(x - cx) \cdot Z}{fx} \quad \text{et} \quad Y = \frac{(y - cy) \cdot Z}{fy}$$

Largeur et hauteur physiques :

$$\text{Largeur} = \sqrt{(X_2 - X_1)^2 + (Z_2 - Z_1)^2}$$

$$\text{Hauteur} = \sqrt{(Y_2 - Y_1)^2 + (Z_2 - Z_1)^2}$$

$$\text{Surface} = \text{Largeur} \times \text{Hauteur}$$

6 Serveur et fichier `app.py`

Le fichier `app.py` contient le code principal qui permet au serveur de fonctionner. Il s'appuie sur le micro-framework web **Flask**, un outil simple et léger permettant de créer des applications web en Python.

1. Qu'est-ce que Flask ?

Flask est un cadre (framework) qui permet de construire des sites ou services web avec du code Python. Il permet notamment de :

— Définir des pages web ou des API accessibles depuis un navigateur ou une application mobile ;
— Recevoir des fichiers envoyés par l'utilisateur (ici, des images dans un fichier ZIP) ;
— Lancer automatiquement des fonctions Python lorsque certaines pages sont appelées ;
— Renvoyer une réponse (soit une page HTML, soit un fichier JSON) après traitement.

2. Rôle de `app.py`

Le fichier `app.py` crée et configure un serveur local. Ce serveur attend des fichiers envoyés par l'application iOS. Lorsqu'un fichier ZIP est reçu :

1. Il est enregistré dans un dossier nommé `uploads/` ;
2. Il est envoyé à une fonction d'analyse (`analyze_zip`) pour traitement ;
3. Une réponse est construite et renvoyée à l'utilisateur : soit une page HTML avec le résultat, soit un objet JSON lisible par l'application.

3. Fonctionnement étape par étape

— **Initialisation du serveur** : l'application Flask est créée avec les dossiers nécessaires (`uploads/` pour les fichiers reçus, `output/` pour les résultats).
— **Accueil** : la page `/` affiche un formulaire web qui permet de téléverser un fichier ZIP manuellement.
— **Envoi via formulaire web (`/upload`)** :
  — L'utilisateur choisit un fichier ZIP et un type de détection (`damage` ou `equipment`) ;
  — Le fichier est enregistré et traité ;
  — Une page HTML avec les résultats (dimensions, type, position) et une image annotée s'affiche.

— **Envoi via l'application mobile ou un client API (`/upload_zip`)** :
  — L'application envoie un ZIP via une requête POST ;
  — Le serveur lit le fichier et appelle la fonction d'analyse ;
  — Une réponse au format `JSON` est renvoyée automatiquement avec toutes les informations :
    — Type détecté (ex : "fissure" ou "prise") ;
    — Dimensions en millimètres (largeur, hauteur, surface) ;
    — Position (ex : "en bas à gauche") ;
    — Lien vers l'image annotée.
— **Téléchargement de fichiers** :
  — Les routes `/uploads/<nom>` et `/output/<nom>` permettent de consulter les fichiers enregistrés et les images annotées.
— **Lancement du serveur** : la commande `app.run()` démarre le serveur local sur le port 5000. Si on utilise `Ngrok`, cela crée un tunnel sécurisé vers Internet.

4. Détection dynamique : dommages ou équipements

L'utilisateur peut indiquer le type de détection souhaité grâce à un champ appelé `detection_type`. Ce paramètre est ensuite transmis à la fonction d'analyse pour adapter les instructions envoyées au modèle Qwen-VL.

5. Exemple simplifié de flux

— L'utilisateur prend une photo avec l'application ;
— Le ZIP est envoyé au serveur via `/upload_zip` ;
— Le serveur traite le fichier, détecte le type d'objet ou dommage ;
— Il calcule la taille, la position, et génère une image annotée ;
— Il renvoie tout cela sous forme de réponse JSON à l'application.

En résumé : le fichier `app.py` transforme le projet en un service web complet. Il automatise la réception des données, leur traitement par intelligence artificielle, et la génération de résultats. Ce serveur fonctionne en continu et constitue le lien entre le terrain (l'application mobile) et le moteur d'analyse Python.

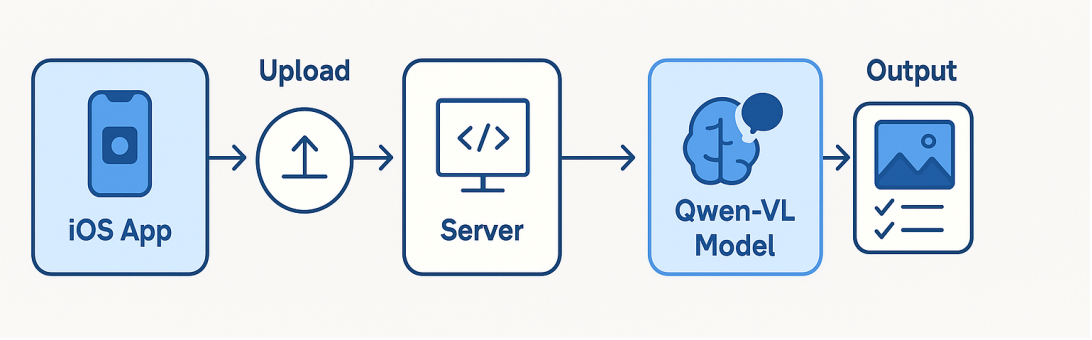
7 Intégration avec une application iOS

Ce projet est conçu pour fonctionner en lien direct avec une application mobile iOS. Celle-ci permet aux utilisateurs (inspecteurs, techniciens, etc.) d'interagir de manière simple et automatisée avec le système d'analyse. Voici le déroulement complet du processus, étape par étape :

1. **Prise de photo**
   L'utilisateur ouvre l'application iOS et prend une photo d'un dommage (fissure, trou, rayure) ou d'un équipement (prise, interrupteur, extincteur).
2. **Génération automatique d'une archive ZIP**
   L'application génère immédiatement une archive au format `.zip` qui contient :
   — `image1.jpg` : la photo prise par l'utilisateur ;
   — `calibration.json` : un fichier contenant des métadonnées essentielles, notamment :
      — `focalLengthX`, `focalLengthY` : longueurs focales de la caméra en pixels ;
      — `principalPointX`, `principalPointY` : coordonnées du centre optique ;
      — `resolutionWidth`, `resolutionHeight` : dimensions de l'image.
3. **Envoi automatique vers le serveur**
   L'application envoie automatiquement cette archive ZIP au serveur distant via une requête HTTP POST. Ce transfert est rapide et ne nécessite aucune action manuelle.
4. **Traitement sur le serveur avec le modèle Qwen-VL**
   Une fois reçu, le serveur décompresse l'archive, lit les paramètres de calibration, et traite l'image grâce au modèle visuel Qwen-VL. Ce dernier détecte, identifie et mesure automatiquement les éléments visibles (dommages ou équipements).
5. **Réponse structurée en JSON**
   Après traitement, le serveur renvoie un fichier JSON contenant toutes les informations utiles :
   — Le type de dommage détecté (ex : `fissure`) ;
   — Les dimensions physiques en millimètres (largeur, hauteur, surface) ;
   — La position approximative dans l'image ;
   — L'URL vers l'image annotée.
6. **Affichage immédiat du résultat**
   L'utilisateur reçoit la réponse directement dans l'application, avec un affichage visuel intuitif (image annotée + données techniques). Le tout est réalisé en quelques secondes.

Ce système permet ainsi une inspection rapide, fiable et autonome, sans connaissance technique particulière.

8 Schéma illustratif



9 Lancement du serveur

Exécution persistante (production légère)

`nohup python app.py > flask.log 2>&1 &`

Cette commande permet de démarrer le serveur qui gère le traitement des images. Voici une explication simple de chaque partie :

— `nohup` : cela signifie “ne pas s’arrêter même si on ferme la fenêtre”. Le serveur reste actif.
— `python app.py` : lance le programme principal du projet.
— `> flask.log` : tout ce que le serveur affiche est enregistré dans un fichier nommé `flask.log`.
— `2>&1` : même les messages d’erreurs sont enregistrés dans ce fichier.
— `&` : le programme s’exécute en arrière-plan, on peut continuer à utiliser l’ordinateur.

Cette commande est idéale si l’on veut que le serveur fonctionne tout seul, même si on quitte la session.

Ouverture au réseau via Ngrok

`nohup ngrok http 5000 --log=stdout > ngrok.log 2>&1 &`

Cette commande permet d'ouvrir un tunnel temporaire vers Internet, pour que l'application mobile puisse envoyer ses photos au serveur. Voici ce qu'elle fait :

— `ngrok http 5000` : crée un lien Internet temporaire qui redirige vers le port `5000`, là où le serveur écoute.
— `nohup`, `>` et `2>&1` : comme ci-dessus, on exécute en arrière-plan et on enregistre les messages dans `ngrok.log`.

Une fois lancée, Ngrok affiche une adresse du type :

`https://abc123.ngrok-free.app`

Cette adresse est utilisée par l'application iOS pour envoyer automatiquement les fichiers ZIP vers le serveur.

10 Structure du projet

```
aws_llmv_ios-main/
 app.py               # Application Flask (backend)
 process_depth.py     # Traitement et analyse
 templates/
    index.html       # Formulaire d'upload
    result2.html     # Résultats visuels
 uploads/             # Images téléversées
 output/              # Images annotées
```

11 Utilisation via l'interface Web

L'application propose une interface Web simple et intuitive accessible via `http://localhost:5000`. Cette interface permet à l'utilisateur de téléverser un fichier `.zip` contenant :

— Une image du dommage ou de l'équipement (`.jpg`, `.png`...).
— Un fichier `calibration.json` avec les paramètres optiques de l'appareil.

11.1 Page d'accueil — Téléversement (`index.html`)

— L'utilisateur sélectionne un fichier `.zip`.
— Il choisit ce qu'il souhaite détecter via un menu déroulant :
  — `Dommages`
  — `Équipements`
— Il clique sur le bouton **Analyser** pour envoyer les données au serveur Flask.

11.2 Affichage des résultats (`result2.html`)

Une fois l'analyse terminée, l'utilisateur est redirigé vers une page de résultats structurée affichant :

— Le type d'objet détecté (ex : `trou`, `prise`, `extincteur`).
— La largeur et la hauteur estimées (en millimètres).
— La surface calculée (en mm^2).
— La position approximative dans l'image (ex : "au centre à droite").
— L'image annotée avec une boîte englobante autour de l'objet détecté.

Un bouton permet également de revenir à la page d'accueil pour analyser un autre fichier.

Analyser un autre fichier

Résultat de l'analyse

Type de détection : Hole
Largeur : 13 mm
Hauteur : 15 mm
Surface : 195 mm²
Position : au milieu au centre

Image traitée