

Mini-Projet :

Vision Artificielle et Traitement des Images

Spécialité : Génie informatique 3

Thème :

Détection des Changements dans les Pièces d'un Appartement à l'Aide d'un Algorithme de Vision Artificielle

Réalisé par :

TANAFAAT Abdessamad

Sous la direction de :

Chaouki Brahim El Khalil

Année Universitaire 2024-2025

Table des matières

Introduction
1 Contexte et Objectif du Projet :
2 Présentation des Technologies Utilisées:.....
Description Générale du Projet
1 Problématique et Besoins à Résoudre
2 Fonctionnalités Développées.....
Traitement des Images.....
1 Chargement et Prétraitement des Images.....
2 Détection de la Région d'Intérêt (ROI:
3 Masquage et Transformation des Images.....
Détection des Objets.....
1 Identification des Différences entre Images
2 Application des Seuils et Morphologie
3 Extraction des Contours et Visualisation
Génération et Sauvegarde des Résultats :
1 Création des PDF de Résultats Intermédiaires
2 Sauvegarde des Résultats Finaux
Démonstration

1. Introduction

1.1 Contexte et Objectif du Projet

Dans le cadre du développement d'applications de vision par ordinateur, l'identification et la détection d'objets dans des environnements spécifiques jouent un rôle essentiel dans divers domaines, tels que la domotique, la sécurité et la gestion des espaces. L'objectif de ce projet est de concevoir une solution automatisée capable de détecter les objets modifiés ou manquants dans des images d'intérieur, tout en générant des rapports visuels et interactifs.

Ce projet se concentre sur trois pièces spécifiques : Chambre, Cuisine, et Salon, avec une approche adaptée à chaque espace pour optimiser la précision de la détection. En utilisant des algorithmes de traitement d'images avancés, l'application vise à :

Identifier les régions d'intérêt (ROI) pour chaque pièce.

Analyser les différences entre une image de référence et une image capturée.

Déetecter les objets ajoutés, supprimés ou déplacés dans l'environnement.

Générer des rapports au format PDF pour une visualisation claire des étapes du processus.

1.2 Présentation des Technologies

Utilisées

Ce projet utilise plusieurs bibliothèques Python puissantes pour traiter et analyser des images. Voici un résumé des technologies principales :

OpenCV : Bibliothèque de traitement d'images qui permet de lire, manipuler et analyser les images, notamment pour la détection d'objets, le calcul des différences entre les images, et la gestion des régions d'intérêt (ROI).

NumPy : Bibliothèque pour la manipulation de données numériques sous forme de tableaux, utilisée ici pour gérer les images et effectuer des calculs mathématiques.

Pillow (PIL) : Bibliothèque pour la manipulation d'images, utilisée pour convertir les images traitées en formats compatibles et les sauvegarder sous forme de fichiers JPEG.

img2pdf : Bibliothèque pour convertir les images en fichiers PDF de haute qualité, permettant de générer des rapports visuels complets des étapes de traitement.

Ces technologies combinées permettent de créer une solution complète pour la détection d'objets dans des images et la génération de rapports sous forme de PDF.

2. Description Générale du Projet

2.1 Problématique et Besoins à Résoudre

l'objectif principal est d'automatiser le processus de détection d'objets dans des images issues de différentes pièces d'une maison, tout en générant un rapport détaillé du processus. Ce système doit répondre aux besoins suivants :

Détection d'Objets Précise : La détection des objets dans les images doit être précise et fiable, en se concentrant sur des régions spécifiques des images, appelées régions d'intérêt (ROI). Ces zones doivent être définies en fonction de la pièce à analyser (par exemple, Chambre, Cuisine, Salon) et les objets doivent être détectés au sein de ces zones avec une sensibilité élevée.

Traitement d'Images Complexes : Les images peuvent contenir des variations de lumière, des bruits et des différences de couleurs. Le projet doit permettre de comparer des images de référence avec des images capturées en utilisant des méthodes de traitement d'images avancées, telles que la conversion en espace de couleurs LAB et HSV, pour obtenir des résultats fiables malgré ces variations.

Visualisation des Étapes de Traitement : Le processus de détection doit être transparent, permettant de visualiser les différentes étapes, comme le calcul des différences entre les images, les seuils appliqués, et les contours détectés. Cette transparence est cruciale pour comprendre et valider le fonctionnement du système.

Génération Automatique de Rapports : À chaque étape du traitement des images, un rapport visuel doit être généré. Ce rapport doit inclure les images de référence, les images traitées, les différences, ainsi que les résultats de la détection des objets. Le rapport final doit être sous forme de fichier PDF, incluant toutes les étapes pour une documentation complète du processus.

Flexibilité pour Différentes Pièces : Le système doit être adaptable et capable de traiter plusieurs types de pièces avec des configurations différentes (par exemple, des objets et des arrangements distincts entre une cuisine et un salon). Cela nécessite une capacité à ajuster les zones d'intérêt et les critères de détection en fonction de la pièce analysée.

2.2 Fonctionnalités Développées

Le projet a permis de développer plusieurs fonctionnalités principales pour la détection d'objets et la génération de rapports à partir d'images :

1. **Définition Dynamique des ROI** : Des régions d'intérêt spécifiques sont définies pour chaque pièce (Chambre, Cuisine, Salon) afin de se concentrer sur les zones pertinentes des images.

2. **Traitement en Espaces de Couleurs** : Les images sont converties en espaces de couleurs LAB et HSV pour mieux gérer les différences de couleur et faciliter la détection des objets.
3. **Calcul des Différences d'Images** : La différence entre l'image de référence et l'image capturée est calculée pour identifier les changements significatifs, en utilisant les canaux LAB et HSV.
4. **Seuil et Filtrage des Différences** : Un seuil est appliqué pour isoler les différences pertinentes, suivi d'opérations morphologiques pour affiner les résultats et éliminer les bruits.
5. **Détection des Contours** : Les contours des objets détectés sont extraits, filtrés par taille et dessinés sur l'image pour visualiser les objets trouvés.

3. Traitement des Images

3.1 Chargement et Prétraitement des Images :

Cette étape consiste à préparer les images avant leur analyse. Voici les principales opérations effectuées :

- **Chargement des Images :**

Les images d'entrée sont chargées à partir de sources spécifiques, comme une caméra ou un répertoire, en utilisant la bibliothèque OpenCV.

- **Redimensionnement :**

Les images sont redimensionnées pour correspondre à une taille standard facilitant les calculs et garantissant une uniformité dans le traitement.

- **Conversion en Échelle de Gris :**

Pour certaines analyses, les images sont converties en niveaux de gris pour simplifier les calculs et réduire la complexité de traitement.

- **Filtrage et Réduction de Bruit :**

Un flou gaussien ou un filtre médian est appliqué pour réduire le bruit et les artefacts visuels, garantissant une détection plus précise.

3.2 Détection de la Région d'Intérêt (ROI) :

La Région d'Intérêt (ROI) permet de limiter le traitement d'image à une zone spécifique contenant les informations pertinentes.

Définition Dynamique de la ROI :

Les coordonnées de la ROI sont définies en fonction de la pièce à analyser (par exemple, Chambre, Cuisine, ou Salon).

Exemple : Une région rectangulaire est définie à partir des points (x, y, largeur, hauteur) spécifiques à chaque pièce.

Isolation de la Zone Pertinente :

Une fois la ROI définie, seule cette région est extraite de l'image pour le traitement ultérieur, économisant ainsi du temps de calcul et augmentant la précision.

3.3 Masquage et Transformation des Images :

Pour améliorer la détection des différences et des objets, des transformations supplémentaires sont effectuées sur les images.

Masquage de la Région Non Pertinente :

Un masque est appliqué pour masquer les zones en dehors de la ROI, rendant ces zones noires. Cela garantit que seuls les pixels à l'intérieur de la ROI sont pris en compte lors de la comparaison.

Transformation en Espaces de Couleurs (LAB et HSV) :

Les images sont converties en deux espaces de couleurs différents pour mieux distinguer les variations de couleur :

LAB : Permet une meilleure séparation des couleurs et de la luminosité.

HSV : Facilite la détection des teintes et saturations dans l'image.

Fusion des Canaux :

Les différences entre les images de référence et les images capturées sont calculées séparément pour chaque canal de couleur, puis fusionnées pour obtenir une image de différence combinée.

4. Détection des Objets

La détection des objets repose sur l'analyse des différences entre les images, l'application de transformations morphologiques et l'extraction des contours. Ces étapes permettent d'identifier les zones d'intérêt et d'afficher les résultats de manière claire.

4.1 Identification des Différences entre Images

L'identification des différences repose sur la comparaison de deux images, souvent une image de référence et une image capturée :

- Soustraction d'Arrière-Plan :

La méthode de soustraction pixel par pixel est utilisée pour détecter les zones de changement. Cela génère une image de différence où les variations entre les deux images sont mises en évidence.

Exemple :

- Utilisation de Canaux de Couleur :

Les différences sont analysées sur des canaux spécifiques (comme LAB ou HSV) pour une meilleure précision.

LUM (Luminosité) : Mise en évidence des zones plus claires ou sombres.

Couleur (Teinte/Saturation) : Détection des variations dans les couleurs des objets.

4.2 Application des Seuils et Morphologie

Couleur (Teinte/Saturation) : Détection des variations dans les couleurs des objets.

Une fois les différences identifiées, des techniques de binarisation et de traitement morphologique sont appliquées pour améliorer la précision :

Seuil Binaire :

Les pixels de l'image de différence sont convertis en noir et blanc en appliquant un seuil.

- Les pixels avec des valeurs supérieures au seuil sont définis comme des **zones d'intérêt (blanc)**, et les autres comme **fond (noir)**.

Opérations Morphologiques :

Ces opérations aident à éliminer le bruit et à raffiner les zones d'intérêt :

- **Dilatation** : Agrandit les zones blanches pour combler les petits trous.
- **Érosion** : Réduit les zones blanches pour supprimer les petits artefacts.
- **Ouverture/Fermeture** : Combine dilatation et érosion pour un nettoyage optimal.

4.3 Extraction des Contours et Visualisation

L'étape finale consiste à extraire et visualiser les contours des objets détectés :

Détection des Contours (Canny) :

L'algorithme de Canny est appliqué pour identifier les contours des objets dans l'image binaire.

Les contours permettent de représenter visuellement les formes et les bords des objets.

Encadrement des Objets :

Les contours détectés sont entourés par des boîtes englobantes (bounding boxes) pour une meilleure compréhension visuelle.

Exemple : Dessiner des rectangles autour des objets détectés à l'aide de OpenCV (cv2.rectangle).

Superposition sur l'Image Originale :

Les contours ou boîtes englobantes sont superposés sur l'image originale pour fournir une visualisation claire des objets détectés.

Ces étapes permettent de transformer les différences d'image en objets clairement identifiés, facilitant l'analyse et l'interprétation des résultats.

5. Génération et Sauvegarde des

Résultats :

La génération et la sauvegarde des résultats sont des étapes cruciales pour documenter et conserver les données issues du traitement. Dans ce projet, ces étapes comprennent la création de rapports intermédiaires et la sauvegarde des données finales pour une utilisation ultérieure ou une analyse approfondie.

5.1 Crédit des PDF de Résultats Intermédiaires

Pour chaque étape intermédiaire du traitement des images, des résultats visuels et descriptifs sont générés et regroupés dans des fichiers PDF :

- **Inclusion des Images Prétraitées :**

Les images obtenues après chaque étape clé, comme le prétraitement, la détection des régions d'intérêt (ROI), ou l'application des seuils, sont ajoutées au rapport PDF pour illustrer les progrès du traitement.

- **Ajout des Métriques et Observations :**

Des informations telles que les seuils utilisés, le nombre d'objets détectés, et les paramètres de transformation sont documentées pour chaque étape. Cela permet une meilleure traçabilité des ajustements effectués au cours du projet.

- **Création Dynamique des Rapports :**

Une bibliothèque comme **reportlab** (en Python) ou des outils similaires est utilisée pour générer les PDF automatiquement :

- Intégration d'images et de légendes.
- Organisation par sections correspondant aux étapes du traitement.

- **Exemple d'Utilisation :**

- Section 1 : Image originale.
- Section 2 : Résultats du masquage et des transformations.
- Section 3 : Objets détectés, avec une boîte englobante et des contours.

5.2 Sauvegarde des Résultats Finaux

Une fois le traitement terminé, les résultats finaux sont sauvegardés sous des formats exploitables et structurés :

- **Images Résultantes :**

Les images finales avec les objets détectés (boîtes englobantes et contours) sont exportées au format **PNG** ou **JPEG** pour une visualisation rapide.

- **Fichiers Structurés :**

Les données liées aux objets détectés, comme leurs coordonnées, tailles, et autres métriques, sont sauvegardées au format **CSV** ou **JSON** pour faciliter une analyse quantitative.

- **Dépôt des Fichiers :**

Les fichiers générés sont organisés dans une hiérarchie claire :

- **Rapports Intermédiaires** : Dans un répertoire dédié (e.g., results/intermediate_reports/).
- **Données Finales** : Dans un répertoire séparé (e.g., results/final_outputs/).

- **Automatisation de la Sauvegarde :**

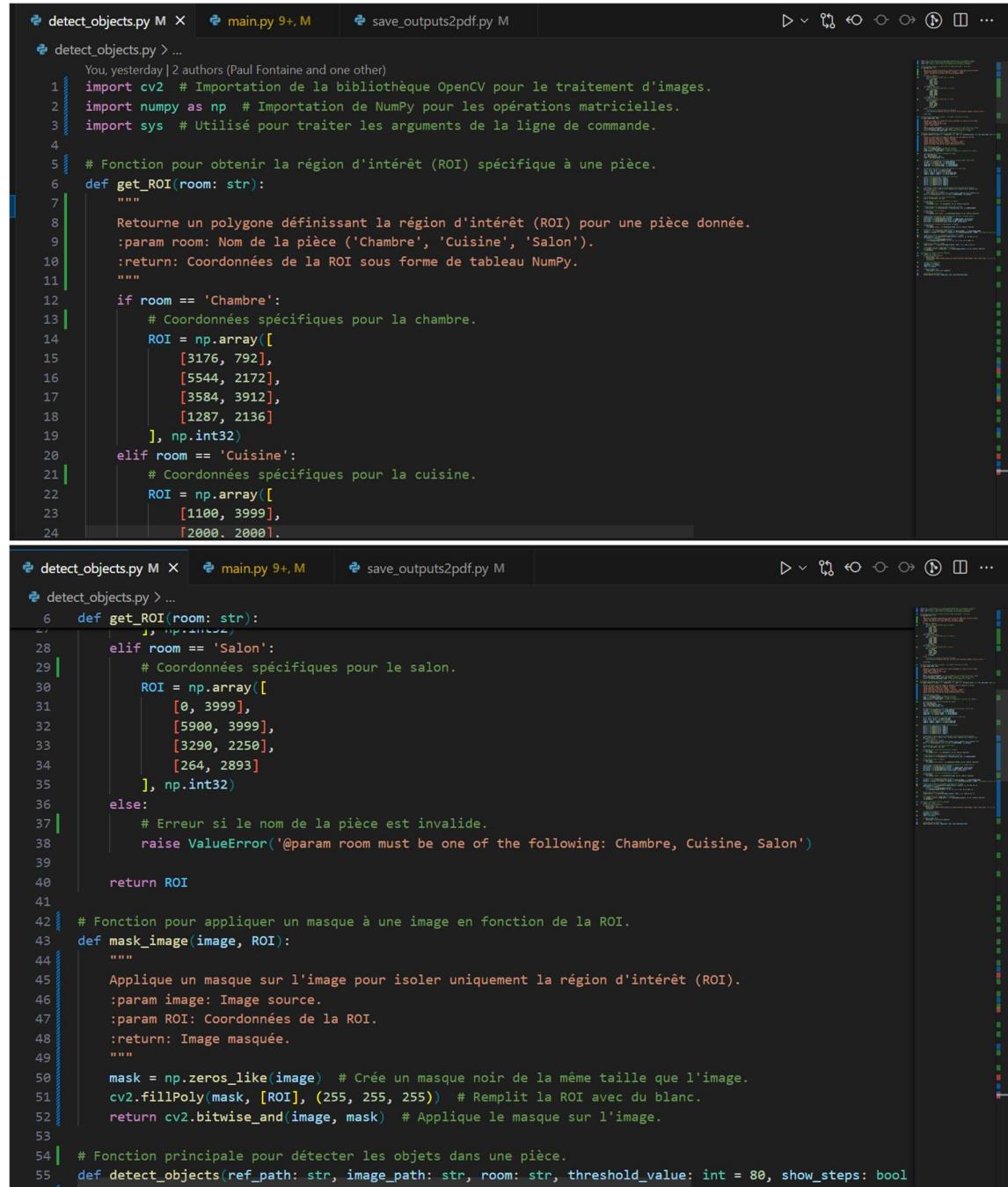
Un script Python est mis en place pour gérer automatiquement la sauvegarde des résultats avec des noms de fichiers pertinents et des horodatages :

6. Démonstration:

La démonstration est l'étape finale qui illustre la mise en œuvre complète des fonctionnalités développées dans le cadre du projet. Elle permet de valider la pertinence des traitements réalisés et de montrer les résultats obtenus de manière visuelle et interactive .

Code : Utilisation de Python.

1. La fonction DetectObjects.py :



```

detect_objects.py M X  main.py 9+, M  save_outputs2pdf.py M
detect_objects.py > ...
You, yesterday | 2 authors (Paul Fontaine and one other)
1 import cv2 # Importation de la bibliothèque OpenCV pour le traitement d'images.
2 import numpy as np # Importation de NumPy pour les opérations matricielles.
3 import sys # Utilisé pour traiter les arguments de la ligne de commande.
4
5 # Fonction pour obtenir la région d'intérêt (ROI) spécifique à une pièce.
6 def get_ROI(room: str):
7     """
8         Retourne un polygone définissant la région d'intérêt (ROI) pour une pièce donnée.
9         :param room: Nom de la pièce ('Chambre', 'Cuisine', 'Salon').
10        :return: Coordonnées de la ROI sous forme de tableau NumPy.
11    """
12    if room == 'Chambre':
13        # Coordonnées spécifiques pour la chambre.
14        ROI = np.array([
15            [3176, 792],
16            [5544, 2172],
17            [3584, 3912],
18            [1287, 2136]
19        ], np.int32)
20    elif room == 'Cuisine':
21        # Coordonnées spécifiques pour la cuisine.
22        ROI = np.array([
23            [1100, 3999],
24            [2000, 2000]
25        ], np.int32)
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

```

```

detect_objects.py M X  main.py 9+, M  save_outputs2pdf.py M
detect_objects.py > ...
6 def get_ROI(room: str):
7     """
8         :param room: Nom de la pièce ('Chambre', 'Cuisine', 'Salon').
9         :return: Coordonnées de la ROI sous forme de tableau NumPy.
10    """
11    if room == 'Salon':
12        # Coordonnées spécifiques pour le salon.
13        ROI = np.array([
14            [0, 3999],
15            [5900, 3999],
16            [3290, 2250],
17            [264, 2893]
18        ], np.int32)
19    else:
20        # Erreur si le nom de la pièce est invalide.
21        raise ValueError('@param room must be one of the following: Chambre, Cuisine, Salon')
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

```

```

detect_objects.py M X main.py 9+, M save_outputs2pdf.py M
detect_objects.py > ...
53
54 # Fonction principale pour détecter les objets dans une pièce.
55 def detect_objects(ref_path: str, image_path: str, room: str, threshold_value: int = 80, show_steps: bool
56     """
57     Déetecte les objets dans une image en comparant avec une image de référence.
58     :param ref_path: Chemin de l'image de référence.
59     :param image_path: Chemin de l'image à analyser.
60     :param room: Nom de la pièce ('Chambre', 'Cuisine', 'Salon').
61     :param threshold_value: Valeur de seuil pour la détection.
62     :param show_steps: Affiche les étapes intermédiaires si True.
63     """
64
65     # Chargement des images.
66     ref = cv2.imread(ref_path) # Image de référence.
67     image = cv2.imread(image_path) # Image actuelle.
68     image_contours = image.copy() # Copie de l'image pour y dessiner les contours.
69
70     # Récupération de la ROI et masquage des images.
71     ROI = get_ROI(room)
72     ref = mask_image(ref, ROI)
73     image = mask_image(image, ROI)
74
75     # Conversion des images en différents espaces colorimétriques (LAB et HSV).
76     ref_LAB = cv2.cvtColor(ref, cv2.COLOR_BGR2LAB)
77     ref_HSV = cv2.cvtColor(ref, cv2.COLOR_BGR2HSV)
78     image_LAB = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
79     image_HSV = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
80
81     # Séparation des canaux (L, A, B pour LAB et H, S, V pour HSV).
82     ref_L, ref_A, ref_B = cv2.split(ref_LAB)
83     ref_H, ref_S, ref_V = cv2.split(ref_HSV)
84     image_L, image_A, image_B = cv2.split(image_LAB)
85     image_H, image_S, image_V = cv2.split(image_HSV)
86
87     # Calcul des différences absolues entre les canaux.
88     diff_L = cv2.absdiff(ref_L, image_L)
89     diff_A = cv2.absdiff(ref_A, image_A)
90     diff_B = cv2.absdiff(ref_B, image_B)
91     diff_H = cv2.absdiff(ref_H, image_H)
92     diff_S = cv2.absdiff(ref_S, image_S)
93     diff_V = cv2.absdiff(ref_V, image_V)
94
95     # Combinaison pondérée des différences pour créer une carte de différences.
96     diff = diff_L * 0.1 + diff_A * 0.3 + diff_B * 0.3 + diff_H * 0.4 + diff_S * 0.2
97     if room == 'Salon':
98         # Ajustement pour le salon.
99         diff = diff * 0.3 + diff_V * 0.3 + diff_S * 0.2 + diff_A * 0.1 + diff_B * 0.1
100    diff = cv2.normalize(diff, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8UC1)
101
102    # Floutage pour lisser la carte de différences.
103    diff = cv2.blur(diff, (25, 25))
104
105    # Affichage de la carte des différences si demandé.
106    if show_steps:
107        cv2.imshow(f'diff', cv2.resize(diff, (0, 0), fx=0.15, fy=0.15))
108
109    # Binarisation de la carte des différences avec un seuil.

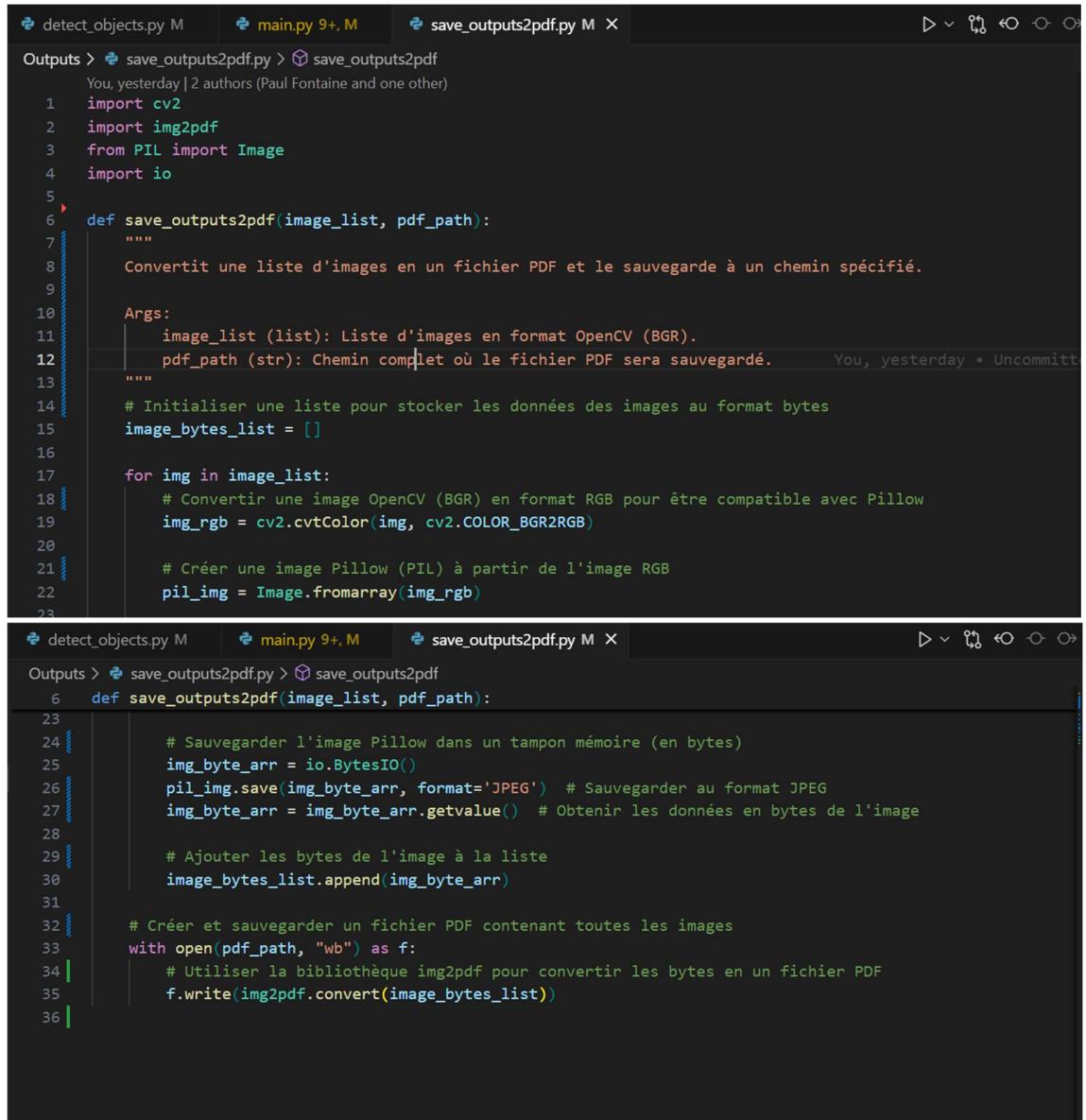
```

```

detect_objects.py M X main.py 9+, M save_outputs2pdf.py M
detect_objects.py > ...
55  def detect_objects(ref_path: str, image_path: str, room: str, threshold_value: int = 80, show_steps: bool
128
129 |     # Dessin des contours et rectangles englobants sur l'image.
130 |     image_contours = cv2.drawContours(image_contours, contours, -1, (0, 255, 0), 4)
131 |     for contour in contours:
132 |         x, y, w, h = cv2.boundingRect(contour)
133 |         cv2.rectangle(image_contours, (x, y), (x + w, y + h), (0, 0, 255), 5)
134
135 |     # Dessin de la ROI sur l'image.
136 |     image_contours = cv2.polyline(image_contours, [ROI], True, (255, 0, 0), 3)
137
138 |     # Affichage final de l'image avec les contours.
139 |     cv2.imshow(f'contours_{image_path}', cv2.resize(image_contours, (0, 0), fx=0.15, fy=0.15))
140 |     cv2.waitKey(0)
141
142 |     # Exécution principale (ligne de commande).
143 |     if __name__ == '__main__':
144 |         # Vérification des arguments.
145 |         if len(sys.argv) < 4:
146 |             print('Usage: python detect_objects.py <path/reference> <path/image> <room> [show_steps : \'0\' or \'1\'')
147 |             sys.exit(1)
148
149 |         # Lecture des paramètres.
150 |         ref_path = sys.argv[1]
151 |         image_path = sys.argv[2]
152 |         room = sys.argv[3]
153
154 |         if len(sys.argv) == 5:
155 |             show_steps = bool(int(sys.argv[4]))
156
157 |             if len(sys.argv) == 5:
158 |                 show_steps = bool(int(sys.argv[4]))
159
159
detect_objects.py M X main.py 9+, M save_outputs2pdf.py M
detect_objects.py > ...
55  def detect_objects(ref_path: str, image_path: str, room: str, threshold_value: int = 80, show_steps: bool
104 |     # Affichage de la carte des différences si demandé.
105 |     if show_steps:
106 |         cv2.imshow(f'diff', cv2.resize(diff, (0, 0), fx=0.15, fy=0.15))
107
108 |     # Binarisation de la carte des différences avec un seuil.
109 |     _, diff_thresh = cv2.threshold(diff, threshold_value, 255, cv2.THRESH_BINARY)
110
111 |     # Affichage du seuil si demandé.
112 |     if show_steps:
113 |         cv2.imshow(f'diff_thresh', cv2.resize(diff_thresh, (0, 0), fx=0.15, fy=0.15))
114
115 |     # Application d'opérations morphologiques pour améliorer le masque binaire.
116 |     kernel_morph = np.ones((25, 25), np.uint8)
117 |     diff_morph = cv2.morphologyEx(diff_thresh, cv2.MORPH_CLOSE, kernel_morph)
118 |     diff_morph = cv2.morphologyEx(diff_morph, cv2.MORPH_OPEN, kernel_morph)
119 |     diff_morph = cv2.dilate(diff_morph, kernel_morph, iterations=2)
120
121 |     # Affichage des résultats morphologiques si demandé.
122 |     if show_steps:
123 |         cv2.imshow(f'morph', cv2.resize(diff_morph, (0, 0), fx=0.15, fy=0.15))
124
125 |     # Détection des contours des objets détectés.
126 |     contours, _ = cv2.findContours(diff_morph, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
127 |     contours = [contour for contour in contours if cv2.contourArea(contour) > 5000] # Filtrage des petits
128
129 |     # Dessin des contours et rectangles englobants sur l'image.
130 |     image_contours = cv2.drawContours(image_contours, contours, -1, (0, 255, 0), 4)
131 |     for contour in contours:
132 |         x, y, w, h = cv2.boundingRect(contour)
133
133

```

2. La fonction save_outputs2pdf.py :



```

Outputs > save_outputs2pdf.py > save_outputs2pdf
You, yesterday | 2 authors (Paul Fontaine and one other)
1 import cv2
2 import img2pdf
3 from PIL import Image
4 import io
5
6 def save_outputs2pdf(image_list, pdf_path):
7     """
8         Convertit une liste d'images en un fichier PDF et le sauvegarde à un chemin spécifié.
9
10    Args:
11        image_list (list): Liste d'images en format OpenCV (BGR).
12        pdf_path (str): Chemin complet où le fichier PDF sera sauvegardé.
13    """
14    # Initialiser une liste pour stocker les données des images au format bytes
15    image_bytes_list = []
16
17    for img in image_list:
18        # Convertir une image OpenCV (BGR) en format RGB pour être compatible avec Pillow
19        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
20
21        # Créer une image Pillow (PIL) à partir de l'image RGB
22        pil_img = Image.fromarray(img_rgb)
23
24    # Sauvegarder l'image Pillow dans un tampon mémoire (en bytes)
25    img_byte_arr = io.BytesIO()
26    pil_img.save(img_byte_arr, format='JPEG') # Sauvegarder au format JPEG
27    img_byte_arr = img_byte_arr.getvalue() # Obtenir les données en bytes de l'image
28
29    # Ajouter les bytes de l'image à la liste
30    image_bytes_list.append(img_byte_arr)
31
32    # Créer et sauvegarder un fichier PDF contenant toutes les images
33    with open(pdf_path, "wb") as f:
34        # Utiliser la bibliothèque img2pdf pour convertir les bytes en un fichier PDF
35        f.write(img2pdf.convert(image_bytes_list))
36

```

3. La fonction main.py :

The screenshot shows a code editor with two tabs open: `main.py` and `save_outputs2pdf.py`. The `main.py` tab is currently active, displaying Python code for image processing tasks. The `save_outputs2pdf.py` tab is visible in the background.

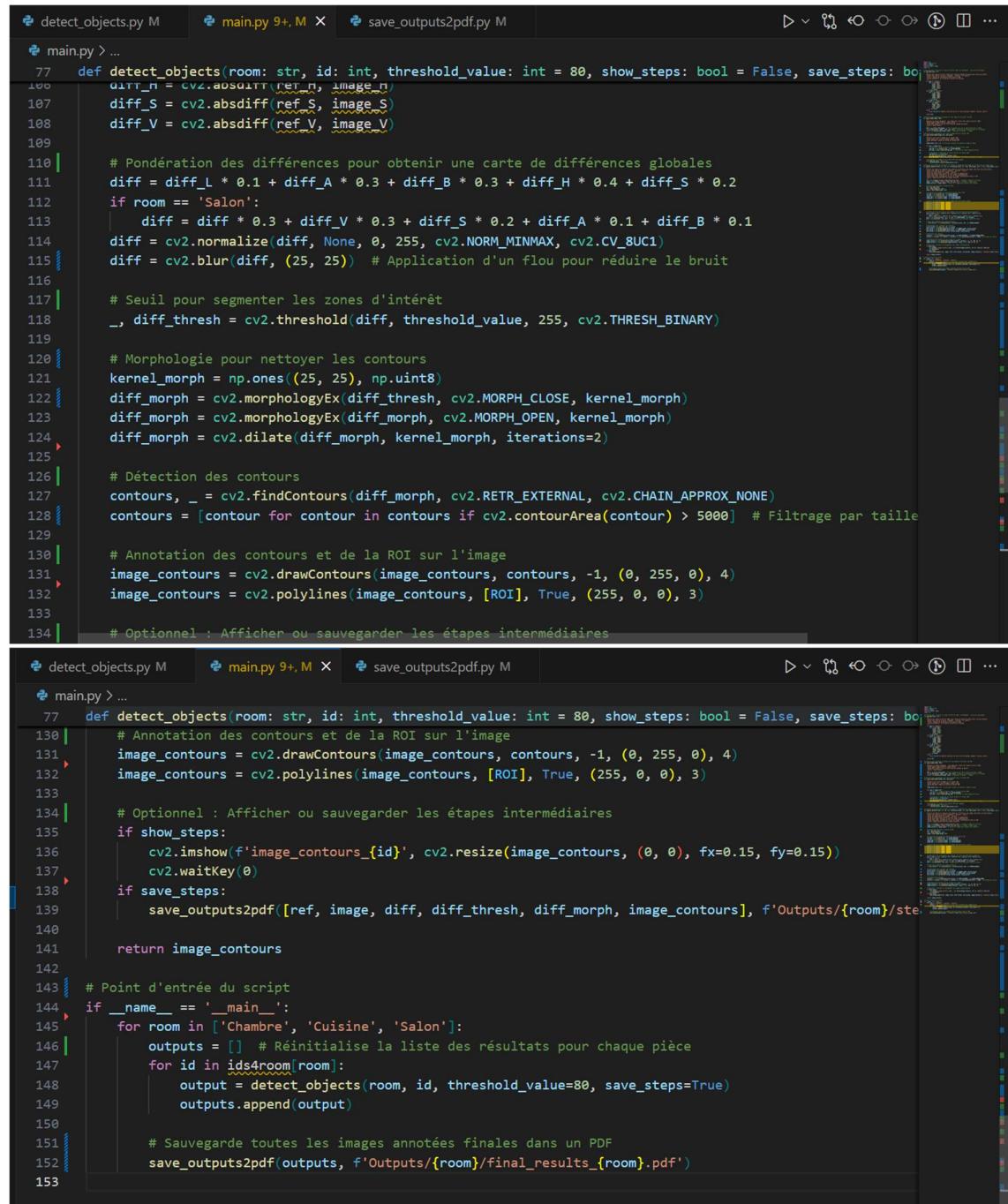
```
You, yesterday | 2 authors (Paul Fontaine and one other)
1 import cv2
2 import numpy as np
3 import img2pdf
4 from PIL import Image
5 import io
6
7 # Fonction pour récupérer la région d'intérêt (ROI) correspondant à une pièce spécifique
8 def get_ROI(room: str):
9     """
10         Renvoie une région d'intérêt (ROI) sous forme de tableau de points pour une pièce donnée.
11         :param room: Nom de la pièce ('Chambre', 'Cuisine', ou 'Salon')
12         :return: Tableau de coordonnées définissant la ROI
13         :raises ValueError: Si le nom de la pièce est invalide
14     """
15     if room == 'Chambre':
16         ROI = np.array([
17             [3176, 7921],
18             [5544, 2172],
19             [3584, 3912],
20             [1287, 2136]
21         ], np.int32)
22     elif room == 'Cuisine':
23         ROI = np.array([
24             [1100, 3999],
25             [2000, 2000],
26             [3600, 2000],
27             [4500, 3999]
28         ], np.int32)
29
30     return ROI
31
32 # Fonction pour appliquer un masque sur une image en utilisant une ROI
33 def mask_image(image, ROI):
34     """
35         Applique un masque polygonal à une image pour isoler une région d'intérêt (ROI).
36         :param image: Image d'entrée (numpy array)
37         :param ROI: Région d'intérêt sous forme de tableau de points
38         :return: Image masquée
39     """
40
41     mask = np.zeros_like(image) # Crée un masque noir de la même taille que l'image
42     cv2.fillPoly(mask, [ROI], (255, 255, 255)) # Dessine le polygone blanc sur le masque
43     return cv2.bitwise_and(image, mask) # Applique le masque à l'image
44
45 # Fonction pour sauvegarder une liste d'images dans un fichier PDF
46 def save_outputs2pdf(image_list, pdf_path):
47     """
```

```

detect_objects.py M main.py 9+, M save_outputs2pdf.py M
main.py > ...

53 # Fonction pour sauvegarder une liste d'images dans un fichier PDF
54 def save_outputs2pdf(image_list, pdf_path):
55 """
56     Convertit une liste d'images en un fichier PDF.
57     :param image_list: Liste d'images (numpy arrays)
58     :param pdf_path: Chemin de sortie du fichier PDF
59 """
60 image_bytes_list = [] # Liste pour stocker les données d'image en bytes
61
62 for img in image_list:
63     # Convertit l'image de OpenCV (BGR) au format Pillow (RGB)
64     img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
65     pil_img = Image.fromarray(img_rgb) # Convertit en objet Image de Pillow
66
67     # Sauvegarde l'image dans un buffer mémoire en format JPEG
68     img_byte_arr = io.BytesIO()
69     pil_img.save(img_byte_arr, format='JPEG')
70     image_bytes_list.append(img_byte_arr.getvalue()) # Ajoute les bytes de l'image
71
72 # Sauvegarde les images en un fichier PDF avec img2pdf
73 with open(pdf_path, "wb") as f:
74     f.write(img2pdf.convert(image_bytes_list))
75
76 # Fonction principale pour détecter les objets en comparant une image de référence et une image cible
77 def detect_objects(room: str, id: int, threshold_value: int = 80, show_steps: bool = False, save_steps: bo
78 """
79     Déetecte les objets en comparant une image de référence et une image cible dans une pièce donnée.
80     :param room: Nom de la pièce ('Chambre', 'Cuisine', 'Salon')
81     :param id: Identifiant de l'image à analyser
82
83     # Fonction principale pour détecter les objets en comparant une image de référence et une image cible
84     def detect_objects(room: str, id: int, threshold_value: int = 80, show_steps: bool = False, save_steps: bo
85 """
86     Déetecte les objets en comparant une image de référence et une image cible dans une pièce donnée.
87     :param room: Nom de la pièce ('Chambre', 'Cuisine', 'Salon')
88     :param id: Identifiant de l'image à analyser
89     :param threshold_value: Valeur du seuil pour la segmentation
90     :param show_steps: Afficher ou non les étapes intermédiaires
91     :param save_steps: Sauvegarder ou non les étapes intermédiaires dans un PDF
92     :return: Image annotée avec les objets détectés
93
94     # Récupère et applique le masque de ROI
95     ROI = get_ROI(room)
96     ref = mask_image(ref, ROI)
97     image = mask_image(image, ROI)
98
99     # Conversion des images en espaces de couleur LAB et HSV
100    ref_LAB = cv2.cvtColor(ref, cv2.COLOR_BGR2LAB)
101    ref_HSV = cv2.cvtColor(ref, cv2.COLOR_BGR2HSV)
102    image_LAB = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
103    image_HSV = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
104
105    # Calcul des différences entre les images dans différents canaux
106    diff_L = cv2.absdiff(ref_L, image_L)
107    diff_A = cv2.absdiff(ref_A, image_A)
108    diff_B = cv2.absdiff(ref_B, image_B)

```



```

detect_objects.py M main.py 9+, M save_outputs2pdf.py M
main.py > ...

77  def detect_objects(room: str, id: int, threshold_value: int = 80, show_steps: bool = False, save_steps: bo
106  diff_L = cv2.absdiff(ref_L, image_L)
107  diff_S = cv2.absdiff(ref_S, image_S)
108  diff_V = cv2.absdiff(ref_V, image_V)
109
110  # Pondération des différences pour obtenir une carte de différences globales
111  diff = diff_L * 0.1 + diff_A * 0.3 + diff_B * 0.3 + diff_H * 0.4 + diff_S * 0.2
112  if room == 'Salon':
113      diff = diff * 0.3 + diff_V * 0.3 + diff_S * 0.2 + diff_A * 0.1 + diff_B * 0.1
114  diff = cv2.normalize(diff, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8UC1)
115  diff = cv2.blur(diff, (25, 25)) # Application d'un flou pour réduire le bruit
116
117  # Seuil pour segmenter les zones d'intérêt
118  _, diff_thresh = cv2.threshold(diff, threshold_value, 255, cv2.THRESH_BINARY)
119
120  # Morphologie pour nettoyer les contours
121  kernel_morph = np.ones((25, 25), np.uint8)
122  diff_morph = cv2.morphologyEx(diff_thresh, cv2.MORPH_CLOSE, kernel_morph)
123  diff_morph = cv2.morphologyEx(diff_morph, cv2.MORPH_OPEN, kernel_morph)
124  diff_morph = cv2.dilate(diff_morph, kernel_morph, iterations=2)
125
126  # Détection des contours
127  contours, _ = cv2.findContours(diff_morph, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
128  contours = [contour for contour in contours if cv2.contourArea(contour) > 5000] # Filtrage par taille
129
130  # Annotation des contours et de la ROI sur l'image
131  image_contours = cv2.drawContours(image_contours, contours, -1, (0, 255, 0), 4)
132  image_contours = cv2.polylines(image_contours, [ROI], True, (255, 0, 0), 3)
133
134  # Optionnel : Afficher ou sauvegarder les étapes intermédiaires
135
136  # Annotation des contours et de la ROI sur l'image
137  image_contours = cv2.drawContours(image_contours, contours, -1, (0, 255, 0), 4)
138  image_contours = cv2.polylines(image_contours, [ROI], True, (255, 0, 0), 3)
139
140  # Optionnel : Afficher ou sauvegarder les étapes intermédiaires
141  if show_steps:
142      cv2.imshow(f'image_contours_{id}', cv2.resize(image_contours, (0, 0), fx=0.15, fy=0.15))
143      cv2.waitKey(0)
144  if save_steps:
145      save_outputs2pdf([ref, image, diff, diff_thresh, diff_morph, image_contours], f'Outputs/{room}/step{id}.jpg')
146
147  return image_contours
148
149
150
151
152
153
# Point d'entrée du script
if __name__ == '__main__':
    for room in ['Chambre', 'Cuisine', 'Salon']:
        outputs = [] # Réinitialise la liste des résultats pour chaque pièce
        for id in ids4room[room]:
            output = detect_objects(room, id, threshold_value=80, save_steps=True)
            outputs.append(output)

        # Sauvegarde toutes les images annotées finales dans un PDF
        save_outputs2pdf(outputs, f'Outputs/{room}/final_results_{room}.pdf')

```

Exécution : Exemple : On choisit par exemple Chambre :

Commande :

```
T:\SemeAnneE\Vision Artificielle\Project\Projet Universitaire>python detect_objects.py Images/Chambre/Reference.JPG
Images/Chambre/IMG_1.JPG Chambre 1
```

Étapes d'Exécution

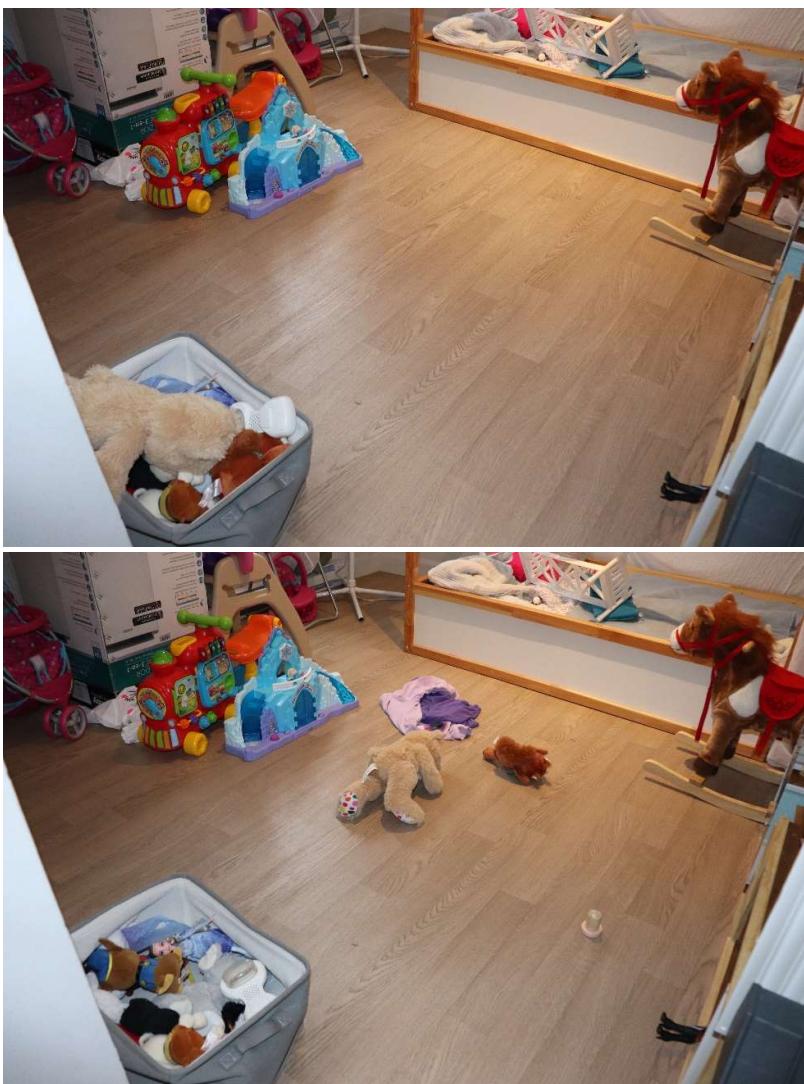
1. Chargement des Images

- o Entrée :

- Reference.JPG : Image de référence.
- IMG_1.JPG : Image à analyser.

- o Action :

- Les deux images sont chargées en mémoire et redimensionnées si nécessaire pour s'assurer de leur compatibilité.



2. Prétraitement des Images

- **Action :**

- Conversion des images en niveaux de gris pour simplifier l'analyse.
- Floutage des images (Gaussian Blur) pour réduire le bruit.

- **Résultat :**

- Les images sont prêtes pour la comparaison.

3. Détection des Différences

- **Action :**

- Soustraction des deux images pour identifier les zones où des changements ont eu lieu.
- Application d'un seuil binaire pour mettre en évidence les régions d'intérêt.

- **Résultat :**

- Une image en noir et blanc montrant les zones différentes.

4. Traitement Morphologique

- **Action :**

- Techniques de dilatation/érosion pour nettoyer les zones détectées (réduction des petits artefacts et amélioration des contours).

- **Résultat :**

- Une image binaire avec des contours nets.

5. Extraction des Contours

- **Action :**

- Détection des contours (algorithme de Canny ou similaire).
- Dessin des contours détectés sur l'image d'origine.

- **Résultat :**

- Une image annotée mettant en évidence les objets ou les zones différentes.

6. Génération de Résultats

