Détection des Changements dans une Scène

Réalisé par : Mathis Aulagnier
Cours : VISION ARTIFICIELLE ET TRAITEMENT DES IMAGES (8INF804)

1 Introduction

L'objectif de ce projet est de détecter des changements dans les environnements intérieurs à travers des images de trois pièces d'un appartement (cuisine, salon, chambre). Le défi principal est de comparer l'état initial de chaque pièce à des images prises dans des conditions variables, où des objets au sol ont été déplacés. Un seul algorithme doit être capable de traiter ces trois scènes, malgré les différences d'éclairage, et de détecter uniquement les changements pertinents, c'est-à-dire ceux au sol, pour ensuite les mettre en valeur à l'aide de *bounding boxes*.

2 Méthodologie

La méthode employée pour détecter les changements dans les scènes d'intérieur repose sur un ensemble d'étapes d'analyse et de traitement d'images. Tout d'abord, chaque image est convertie dans l'espace de couleur LAB, ce qui permet d'extraire la composante L correspondant à la luminosité. Cela réduit l'impact des variations d'éclairage dans les différentes pièces. Ensuite, cette composante L est normalisée à l'aide de l'algorithme CLAHE (Contrast Limited Adaptive Histogram Equalization), qui améliore le contraste de l'image tout en limitant l'amplification du bruit. Un flou gaussien est également appliqué pour atténuer le bruit résiduel. Après le prétraitement, la différence absolue entre l'image de référence et l'image teste est calculée afin d'identifier les zones de changement. Un seuillage est ensuite utilisé pour créer une image binaire, permettant de localiser les changements potentiels. Les contours des objets détectés dans cette image binaire sont extraits, et un filtre basé sur l'aire des objets est appliqué pour éliminer les petits bruits indésirables. Les objets de taille suffisante sont conservés et les contours sont convertis en rectangles. Une méthode de fusion est ensuite employée pour regrouper les rectangles qui se chevauchent, ce qui permet de ne pas multiplier les détections autour d'un même objet. Cependant, lorsque plusieurs objets mal rangés sont proches les uns des autres, l'algorithme tend à les englober dans un seul grand rectangle plutôt que de les détecter individuellement. Afin de se concentrer uniquement sur les objets au sol, un filtrage basé sur la position des objets est appliqué, éliminant ceux situés aux bords ou dans la partie supérieure de l'image. Finalement, les changements pertinents sont mis en valeur par des *bounding boxes*, encadrant ainsi les objets déplacés et permettant une visualisation claire des différences entre l'état initial et l'état actuel de chaque pièce.

3 Difficultés rencontrées

Durant ce projet, j'ai été confronté à plusieurs difficultés techniques liées à la gestion de la luminosité, du contraste et du bruit résiduel dans les images.

3.1 Les espaces de couleurs

Initialement, j'ai commencé à travailler avec des images converties en niveaux de gris. Cependant, cette approche a montré ses limites face aux variations de luminosité, notamment lors de l'ouverture du rideau du salon, ce qui rendait la détection mauvaise. Pour résoudre ce problème, je me suis tourné vers les espaces de couleurs. J'ai d'abord choisi l'espace HSV, souvent recommandé pour la détection d'objets comme mentionné dans le cours.

Mon idée initiale était d'éliminer la composante V (brightness) pour éviter les problèmes de luminosité. Désormais, j'avais le choix entre les composantes H (Hue, teinte) et S (saturation). J'ai décidé de conserver la composante H, car la composante S montrait trop de variations. Malheureusement, cette approche n'a pas donné de résultats satisfaisants.

J'ai donc tenté de fusionner la composante V (luminosité) de l'image de référence avec les composantes H et S de l'image à analyser, mais cela a entraîné une "fusion" des objets, rendant ainsi la détection des contours difficile. J'ai finalement opté pour la conservation exclusive de la composante V dans les deux images (référence

et test). Pour assurer une comparaison adéquate, j'ai normalisé les histogrammes de ces deux images à l'aide de cv2.normalize(...).

Pour améliorer davantage les résultats, j'ai appliqué la normalisation à l'aide de l'algorithme CLAHE, comme recommandé dans le cours. Cet algorithme améliore le contraste en réalisant une égalisation d'histogramme sur de petites régions de l'image, tout en limitant le bruit. Cette approche a considérablement optimisé les résultats.

Je me suis ensuite tourné vers l'espace LAB, où la composante L représente la luminosité. En récupérant la composante L des images de référence et de test, j'ai observé des résultats légèrement meilleurs, ce qui m'a poussé à rester dans cet espace de couleur.

3.2 Réglage fin des paramètres

Une autre difficulté a été de régler les paramètres des différentes opérations de traitement d'image pour obtenir une détection optimale. Les principaux paramètres ajustés étaient :

- `cv2.createCLAHE(clipLimit=7, tileGridSize=(6, 6))` pour contrôler la limite de contraste et la taille de la grille.
- `cv2.GaussianBlur(equalized_image, (5, 5), 0)` pour lisser l'image tout en préservant les contours.
- `cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, np.ones((5,5), np.uint8))` et `cv2.morphologyEx(thresh, cv2.MORPH_OPEN, np.ones((5,5), np.uint8))` pour réduire le bruit dans les contours détectés.
- `cv2.threshold(diff, 55, 255, cv2.THRESH_BINARY)` pour ajuster le seuil de binarisation des différences entre les images.

Ces ajustements ont nécessité plusieurs itérations pour trouver la combinaison qui donnait les résultats les plus précis tout en limitant le bruit.

3.3 Gestion des rectangles de détection

Enfin, un autre défi a été la gestion des rectangles de détection. Plusieurs rectangles étaient souvent générés pour un seul objet déplacé, ou certains rectangles apparaissaient de manière erronée. Pour corriger cela, j'ai mis en place un algorithme qui fusionne les rectangles qui se chevauchaient. En plus de la fusion, je supprime également les rectangles situés à proximité des bords latéraux et de la partie supérieure de l'image. Cette approche a permis de focaliser la détection sur la zone centrale et inférieure de l'image, où les objets d'intérêt étaient situés dans nos différents tests.

4 Résultats et conclusion

Les performances de l'algorithme développé pour la détection de changements dans les scènes de pièces avec des conditions d'éclairage variées ont été satisfaisantes. Grâce aux méthodes de prétraitement, comme l'utilisation des espaces de couleurs adaptés (HSV et LAB) et la normalisation via CLAHE, l'algorithme a pu efficacement compenser les différences de luminosité. Il montre cependant des faiblesses lorsque les images sont sous-exposées.

En termes de résultats quantitatifs, l'algorithme a correctement détecté 51 objets mal rangés sur les 62, soit une précision de 82.0%. Sur les 62 objets mal rangés dans les différentes scènes, 11 n'ont pas été détectés, et 7 fausses détections (changements détectés à tort) ont été enregistrées. Malgré ces erreurs, l'algorithme a montré une bonne capacité à encadrer les objets déplacés ou mal rangés dans les scènes, tout en filtrant les changements non-pertinents.

En conclusion, ce projet m'a permis de développer un programme pour la détection de changements dans des environnements soumis à des variations d'éclairage. Cependant, avec des perspectives d'amélioration future, je pourrais pour augmenter la précision et la robustesse de l'algorithme.

Annexe : Code Python

Voici le code Python utilisé dans le cadre de ce projet :

```
import cv2
import numpy as np

def preporcess_image(image):
    '''
    Cette fonction prend une image en entr e et retourne une image pr trait e.
    '''
    # Convertir l'image en LAB
    lab_img = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    # S parer les canaux
    l, a, b = cv2.split(lab_img)

    #hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    #h, s, v = cv2.split(hsv_img)


    # Appliquer l' galisation d'histogramme sur le canal L avec Clahe jug plus efficace que l' galisation d'histogramme classique
    clahe = cv2.createCLAHE(clipLimit=7, tileGridSize=(6, 6))
    equalized_image = clahe.apply(l)

    # Appliquer un flou gaussien pour r duire le bruit
    blurred_image = cv2.GaussianBlur(equalized_image, (5, 5), 0)

    # Normaliser la composante L pour limiter l'influence de la luminosit
    norm_image = cv2.normalize(blurred_image, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

    return norm_image

def encadrer(diff, img_shape):
    '''
    Cette fonction prend une image de diff rence et la forme de l'image originale en entr e.
    Elle retourne une liste de rectangles encadrant les objets d tect s.
    '''
    # Appliquer un seuillage pour obtenir une image binaire
    _, thresh = cv2.threshold(diff, 55, 255, cv2.THRESH_BINARY)

    # Appliquer une fermeture et une ouverture morphologique pour liminer le bruit
    kernel = np.ones((5,5), np.uint8)
    thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
    thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)

    # Trouver les contours
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Filtrer les contours en fonction de leur aire
    min_area = 4000
    filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) > min_area]

    # Trier les contours par aire d croissante
    sorted_contours = sorted(filtered_contours, key=cv2.contourArea, reverse=True)

    # Garder les 15 plus grands contours
    top_contours = sorted_contours[:15]

    # Convertir les contours en rectangles
    rectangles = [cv2.boundingRect(cnt) for cnt in top_contours]

    # Fusionner les rectangles qui se chevauchent
    merged_rectangles = merge_rectangles(rectangles)

    # Filtrer les rectangles bas s sur leur position
    filtered_rectangles = filter_rectangles(merged_rectangles, img_shape)

    return filtered_rectangles


```

```
69 def merge_rectangles(rectangles):
70     '''
71     Cette fonction prend une liste de rectangles en entr e et retourne une liste de rectangles
        fusionn s.
72     '''
73     if not rectangles:
74         return []
75 
76     merged = []
77     for rect in rectangles:
78         # Si la liste des rectangles fusionn s est vide, ajouter le rectangle actuel
79         if not merged:
80             merged.append(rect)
81         # Sinon, fusionner le rectangle actuel avec un rectangle existant s'ils se che
82         else:
83             merged_rect = rect
84             for i, existing_rect in enumerate(merged):
85                 # Si les rectangles se chevauchent, les fusionner
86                 if rectangles_overlap(merged_rect, existing_rect):
87                     merged_rect = merge_two_rectangles(merged_rect, existing_rect)
88                     merged[i] = merged_rect
89                     break
90             else:
91                 merged.append(merged_rect)
92     return merged
93 
94 def rectangles_overlap(rect1, rect2):
95     '''
96     Cette fonction prend deux rectangles en entr e et retourne True s'ils se chevauchent,
       False sinon.
97     '''
98     x1, y1, w1, h1 = rect1
99     x2, y2, w2, h2 = rect2
100    return not (x1 + w1 < x2 or x2 + w2 < x1 or y1 + h1 < y2 or y2 + h2 < y1)
101 
102 def merge_two_rectangles(rect1, rect2):
103     '''
104     Cette fonction prend deux rectangles en entr e et retourne un rectangle fusionn .
105     '''
106     x1, y1, w1, h1 = rect1
107     x2, y2, w2, h2 = rect2
108     x = min(x1, x2)
109     y = min(y1, y2)
110     w = max(x1 + w1, x2 + w2) - x
111     h = max(y1 + h1, y2 + h2) - y
112     return (x, y, w, h)
113 
114 def filter_rectangles(rectangles, img_shape):
115     '''
116     Cette fonction prend une liste de rectangles et la forme de l'image en entr e.
117     Elle retourne une liste de rectangles filtr s en fonction de leur position.
118     '''
119     height, width = img_shape[:2]
120     left_threshold = width * 0.2
121     right_threshold = width * 0.9
122     top_threshold = height * 0.2
123 
124     filtered = []
125     for rect in rectangles:
126         x, y, w, h = rect
127         if x > left_threshold and x + w < right_threshold and y + h > top_threshold:
128             filtered.append(rect)
129     return filtered
130 
131 def main(reference_path, current_path):
132     '''
133     Cette fonction prend les chemins des images de r f rence et actuelle en entr
134     Elle affiche les diff rences entre les deux images et encadre les objets d tect s.
135     '''
136     # Lire les images
137     reference = cv2.imread(reference_path)
138     current = cv2.imread(current_path)
139 
```

```
    # Pr traiter les images
    reference_eq = preporcess_image(reference)
    current_eq = preporcess_image(current)

    # Calculer la diff rence entre les images
    diff = cv2.absdiff(reference_eq, current_eq)

    # Encadrer les objets d tect s
    changes = encadrer(diff, current.shape)

    # Afficher les objets d tect s
    result = current.copy()  # Use the original image for visualization

    # Dessiner les rectangles autour des objets d tect s
    for rect in changes:
        x, y, w, h = rect
        cv2.rectangle(result, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Afficher l'image r sultante
    cv2.imshow('Detected Changes', result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Listing 1: Code python