



Université du Québec
à Chicoutimi

TP #1 : 8INF804 : Traitement numérique des images - Hiver 2022

Réalisé par :

Nihal OUHAOUADA | OUHN11629909

Pierre LACLAVERIE | LACP03119904

Thibaud SIMON | SIMT15039901

Yann REYNAUD | REYY07110005

Contexte :

Afin d'aider les personnes âgées qui souffrent de négligence de soi à se motiver pour ranger leur domicile, une solution a été conçue pour ce faire. Il s'agit d'un système qui permet de détecter le taux d'encombrement dans une pièce.

Objectif :

Le but de ce TP est de détecter, à l'aide d'un seul algorithme, les potentiels obstacles au sol dans les différentes pièces.

Démarche :

Différentes étapes, visibles dans le fichier main.py de notre programme, permettent d'obtenir le résultat souhaité : encadrer les obstacles détectés afin de générer un score de dangerosité pour l'habitant.

1. Différence entre deux images:

Plusieurs étapes composent le fait d'effectuer notre différence entre images: (difference_filtered.py)

1. Egalisation lumineuse

Les images BGR d'entrée (référence et comparée) sont chargées puis, à l'aide de la fonction equalizeHist de opencv, la luminance des deux images dans un espace de couleur LAB est égalisée. Ainsi, on ignore les différences de luminosité peu marquées.

2. Floutage gaussien

On floute ensuite les deux images pour nettoyer le potentiel bruit présent sur les photographies, avec un kernel (noyau) gaussien de taille 7x7. Ceci aide à faire un prétraitement pour ne garder que l'essentiel de la différence.

3. Niveau de gris

Les images sont converties en niveaux de gris pour avoir une différence uniquement sur la luminance de l'image (variation de couleur non importantes)

4. Différence absolue:

La différence est faite avec cv2.absdiff(ref, comp), qui nous donne une image de retour nuancée entre noir pour zone similaires, et blanc pour zones différentes.

5. A l'exemple du cours, la différence retournée est débruitée de nouveau avec un filtre bilatéral.



2. Application d'un adaptative thresholding

Un thresholding adaptatif est ensuite appliqué à l'image, pour ne garder que les valeurs importantes: les objets qui ont été ajoutés ou enlevés à la scène.



3. Application de quelques transformations morphologiques:

Nous avons appliqué la dilatation et l'érosion pour affiner les zones importantes du résultat du thresholding. Après le passage de l'adaptive thresholding, il reste des fragments non visibles à l'œil, ceux-ci peuvent être source de complications pour la suite des traitements, c'est pour cela que nous utilisons l'ouverture morphologique afin de supprimer les objets de petite taille.

Erosion : érode les formes pour éliminer les petits objets



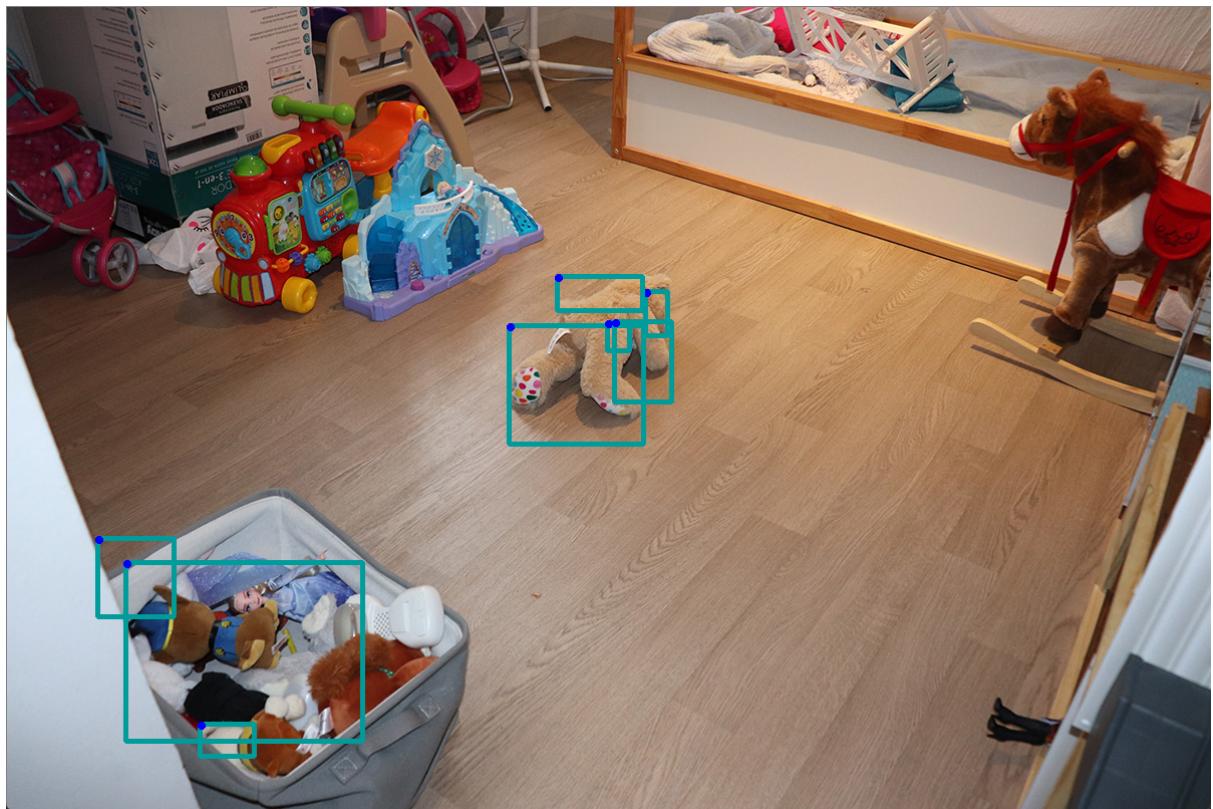
Dilatation: remet en valeur les zones non érodées

Tous les petits objets ayant été éliminés, on va pouvoir dilater les objets afin de reconstruire partiellement les objets.



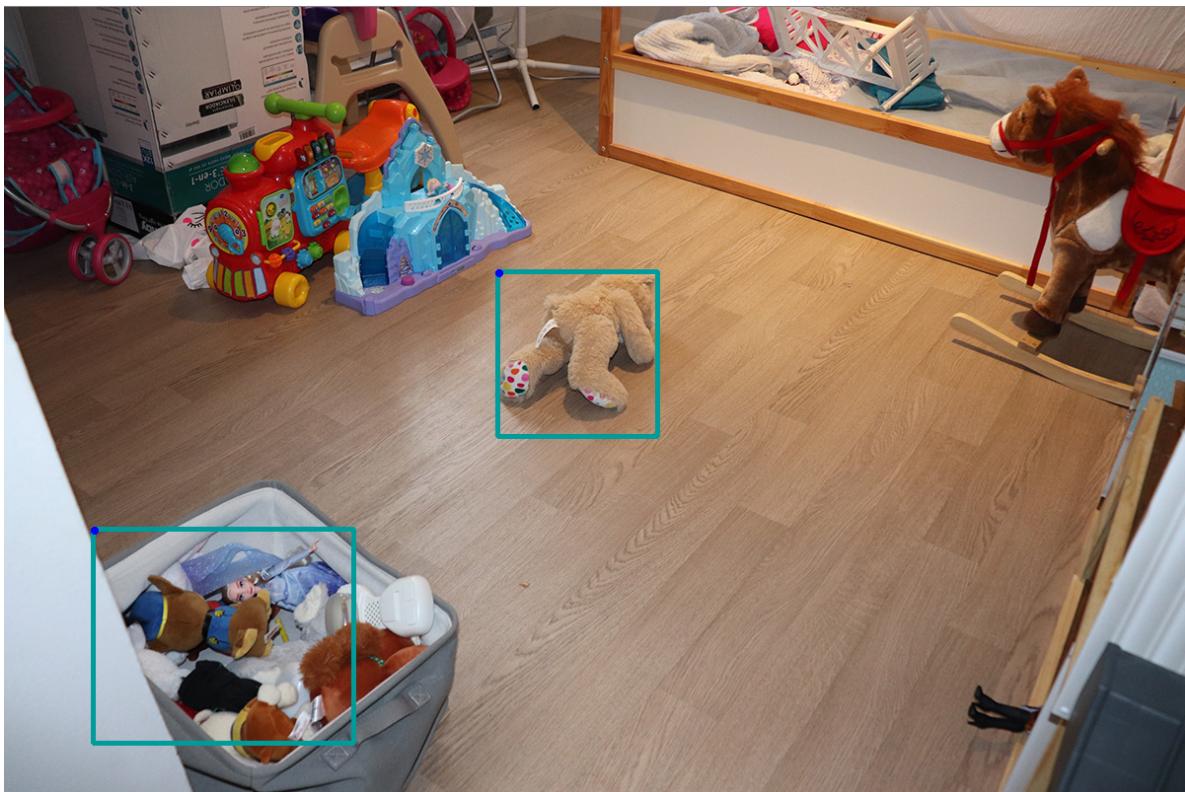
4. Contours bounding box

Les contours sont trouvés à partir de la fonction opencv: cv2.findContours() . Ils sont ainsi dessinés sous forme de rectangles sur l'image grâce à la fonction cv2.DrawRectangle() .



5. Fusion des bounding box imbriquées :

Une méthode développée par nos soins permet que pour tout rectangle qui intersecte un autre rectangle, un rectangle englobant les deux rectangles précédents est créé.



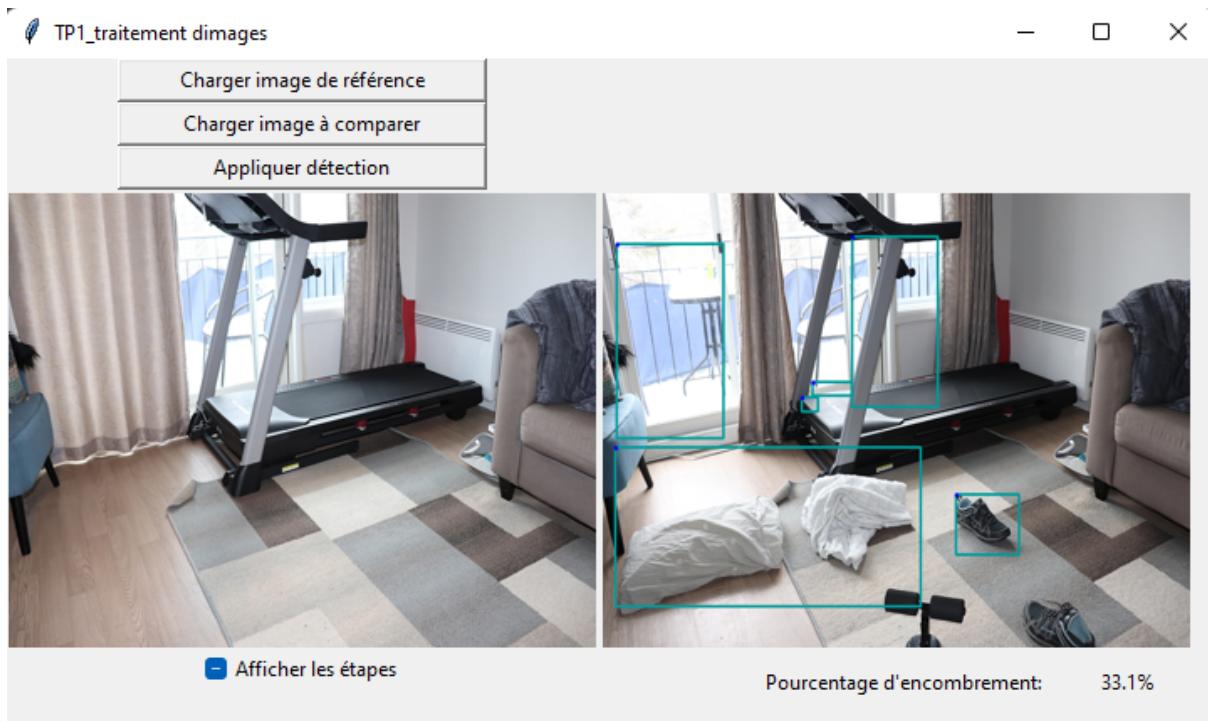
6. Calcul du taux d'encombrement dans la pièce:

Un ratio est calculé à partir de l'aire couverte par les nouveaux rectangles. Pour ce faire, nous avons choisi de faire la somme de l'aire couverte par les contours externes divisée par l'aire de l'image. Il a été préférable pour nous d'opter pour cette solution plutôt que de faire la somme des contours internes (présents par exemple lors d'empilement d'objets), car cette dernière technique aurait pu entraîner des défauts de précision en dédoublant certaines aires lorsque des contours sont imbriqués les uns sur les autres.

Ce ratio multiplié par 100 donne un pourcentage permettant au client d'estimer la surface de la pièce qui est encombrée.



Même image = aucun encombrement



7. Interface:

La dernière étape consistait à implémenter les méthodes déjà étudiées dans une interface graphique qui lis et affiche les images, réalise les différents traitements sur les images et détecte les changements.



La case à cocher “Afficher les étapes” permet d’afficher une fenêtre de rendu pour chaque étape du traitement, lorsque cochée (non cochée par défaut)

Guide d'utilisation:

Le code se compose principalement de 4 fichiers python:

imutils.py : contient les fonctions qui concernent la détection des formes et le dessin des bords sur les images.

difference.py: contient la fonction qui renvoie la différence entre 2 images ainsi que l'égaliseur des 2 images

interface.py: définit la classe et les méthodes de l'interface utilisateur.

main.py: illustre les différentes étapes de notre algorithme et crée la fenêtre.

Une fois l'interface affichée (lancer main), il faut charger l'image de référence ainsi que celle que nous cherchons à comparer à l'aide des deux boutons “charger image de référence” et “charger image à comparer”.

Pour appliquer la détection sur la 2ème image à comparer, il suffit de cliquer sur “Appliquer détection”.

Difficultés rencontrées et améliorations possibles:

Nous avons rencontré plusieurs difficultés au cours de ce TP.

Nous avons par exemple été confrontés à un problème de contour des objets à cause de leur ombre qui avait une influence sur l'intensité des pixels et donc la détection.

De même, nous avons éprouvé quelques difficultés pour différencier un objet du sol lorsque leurs couleurs sont similaires.

Il a aussi fallu trouver des paramètres (thresholding, érosion...) qui fonctionnaient au mieux, même si on changeait d'environnement. Notre approche manuelle n'est pas encore idéale mais elle donne des résultats qui nous semblent corrects.

Nous n'avons pas eu l'occasion d'utiliser des indicateurs quantitatifs pour nous assurer de l'efficacité de notre programme. Un F1 Score couplé d'une matrice de confusion auraient pu nous rapprocher de plus d'efficacité. Nous ne l'avons pas réalisé par souci de la définition des "vrais" et "faux". Utiliser le fait d'avoir un "bon" nombre de rectangles, placés au "bon" endroit, de la "bonne" taille sont des critères que l'on aurait pu prendre, mais ils ne sont pas rigoureux.

Autre approche envisagée:

Pour résoudre cette problématique, plusieurs approches existent. Nous avons choisi de vous présenter celle qui nous semblait la plus adaptée. Durant nos phases de recherche nous avons aussi voulu nous rapprocher des opérateurs de traitement d'images présents dans l'environnement Matlab.

Pour cela, nous avons recréé des fonctions telles que imfill, imclearBorder ou encore bwareaopen. D'autres approches ont aussi utilisé le rehaussement de contraste en image niveaux de gris afin de bien mettre en évidence les objets qui possèdent une couleur similaire au sol. Un exemple est l'ourson en peluche dans la chambre. Nous avons décidé de garder l'approche présentée au long de ce sujet car l'utilisation de ces techniques est moins naturelle que celle présentée tout au long du rapport.

Références :

imfill : <https://www.mathworks.com/help/images/ref/imfill.html>

imclearborder :

https://www.mathworks.com/help/images/ref/imclearborder.html?s_tid=doc_ta

bwareaopen :

https://www.mathworks.com/help/images/ref/bwareaopen.html?s_tid=doc_ta