

Projet Vision par ordinateur

Détection de panneaux de circulation

THIERRY Louis

XU TianCi

NJAMBE MBAPPE Samuel Stéphane

Sommaire

Contexte et objectif du projet	3
Etat de l'art.....	3
Explication de la méthode retenue	4
Résultats obtenus : code source et images obtenues.....	5
Code source Python.....	5
Images obtenues	9
Détection de Panneaux STOP.....	9
Détection de panneaux 30 et ralentisseur	9
Détection de panneaux 50	10
Détection de panneaux Passage piéton	11
Détection de panneaux de Rondpoint	11
Images de référence pour la détection de panneaux	12

Contexte et objectif du projet

L'objectif est de concevoir un programme sous Python de reconnaissance de panneaux de circulation pour voiture autonome.

Nous souhaitons identifier les panneaux de circulation d'une vidéo prise en voiture autour de l'ESIGELEC.

Etat de l'art

1. YOLO (You Only Look Once):

YOLO divise l'image en une grille et prédit les boîtes englobantes ainsi que les probabilités de classe pour chaque cellule de la grille. Il est rapide et peut détecter plusieurs objets sur une image entière. Il est souvent utilisé pour des applications nécessitant une détection d'objets en temps réel, comme la surveillance vidéo et la conduite autonome.

2. SIFT (Scale-Invariant Feature Transform):

SIFT identifie les points d'intérêt dans une image qui sont robustes aux changements d'échelle, de rotation et d'illumination. SIFT extrait des descripteurs locaux autour de ces points d'intérêt pour la reconnaissance d'objets et la correspondance d'images. Il est plus lent que YOLO mais plus robuste dans certaines situations, en particulier lorsqu'il y a des variations d'échelle et de perspective.

3. ORB (Oriented FAST and Rotated BRIEF):

ORB combine les avantages de FAST pour la détection rapide de points d'intérêt et de BRIEF pour la description des caractéristiques. Il est moins coûteux en termes de calcul par rapport à SIFT, ce qui le rend plus adapté aux applications nécessitant une exécution rapide. Il est souvent utilisé dans des applications de correspondance d'images en temps réel et de localisation d'objets.

Explication de la méthode retenue

Nous avons utilisé la méthode de **SIFT** pour le projet.

Explication du processus de détection avec l'exemple de la détection du panneau STOP :

Le programme compare l'image de référence avec la vidéo et cherche des points similaires.

On peut voir sur l'image ci-dessous à quoi ressemble la détection de points :



Le processus consiste à déterminer l'endroit de l'image où il y a le plus grand nombre de points similaires. Ensuite, une fonction sera mise en place pour éliminer les coordonnées erronées, et la moyenne des coordonnées sera calculée à partir des points restants afin de déterminer avec précision la position du panneau. Cette position sera ensuite encadrée par un rectangle et étiquetée. On obtient le résultat ci-dessous :



Résultats obtenus : code source et images obtenues

Code source Python

```
# Importation des bibliothèques nécessaires

import numpy as np
import cv2 as cv
from collections import defaultdict

# Fonction pour calculer la distance euclidienne entre deux points
def distance(coord1, coord2):
    return ((coord1[0] - coord2[0]) ** 2 + (coord1[1] - coord2[1]) ** 2) ** 0.5

# Fonction pour supprimer les valeurs aberrantes d'une liste de coordonnées
def remove_outliers(coordinates, threshold, min_points=15):
    grouped_coordinates = defaultdict(list)

    # Grouper les coordonnées similaires
    for coord in coordinates:
        added = False
        for center_coord, similar_coords in grouped_coordinates.items():
            if any(distance(coord, similar_coord) <= threshold for similar_coord in similar_coords):
                grouped_coordinates[center_coord].append(coord)
                added = True
                break
        if not added:
            grouped_coordinates[tuple(coord)] = [coord]

    # Sélectionner le groupe le plus grand s'il contient suffisamment de points
    if len(grouped_coordinates) > 0:
        max_group = max(grouped_coordinates.values(), key=len)
        if len(max_group) >= min_points:
            cleaned_coordinates = max_group
        else:
            cleaned_coordinates = []
    else:
        cleaned_coordinates = []

    return cleaned_coordinates

# Fonction pour calculer la moyenne des coordonnées dans une liste
def average_coordinates(coordinates):
    x_sum = 0
    y_sum = 0
    for coord in coordinates:
        x_sum += coord[0]
        y_sum += coord[1]
    x_avg = x_sum / len(coordinates)
    y_avg = y_sum / len(coordinates)
    return [x_avg, y_avg]
```

```
# Ouvrir la capture vidéo et charger les images modèles
vid = cv.VideoCapture('Route30s.mp4')
img1 = cv.imread('stop2.png', cv.IMREAD_GRAYSCALE)
img2 = cv.imread('panne.png', cv.IMREAD_GRAYSCALE)
img3 = cv.imread('panneau.png', cv.IMREAD_GRAYSCALE)
img4 = cv.imread('pass.png', cv.IMREAD_GRAYSCALE)
img5 = cv.imread('RP.png', cv.IMREAD_GRAYSCALE)

# Obtenir les images par seconde de la vidéo
fps = vid.get(cv.CAP_PROP_FPS)

# Initialiser le détecteur de caractéristiques SIFT et le comparateur Brute-Force
sift = cv.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)
kp3, des3 = sift.detectAndCompute(img3, None)
kp4, des4 = sift.detectAndCompute(img4, None)
kp5, des5 = sift.detectAndCompute(img5, None)
bf = cv.BFMatcher()

# Variable pour stocker la dernière position détectée
last_position = None

# Boucle à travers chaque frame de la vidéo
while(1):

    # Lire une frame de la vidéo
    ret, frame = vid.read()
    if not ret:
        break

    # Convertir la frame en niveaux de gris et extraire le tiers droit de la frame
    framegray = cv.cvtColor(frame, cv.COLOR_RGB2GRAY)
    height, width = framegray.shape
    right_third_frame = framegray[:, (2*width)//3:]

    # Détecter les points clés et les descripteurs dans la frame
    kp_frame, des_frame = sift.detectAndCompute(right_third_frame, None)

    # Listes pour stocker les points clés correspondants pour chaque image modèle
    matched_pts1 = []
    matched_pts2 = []
    matched_pts3 = []
    matched_pts4 = []
    matched_pts5 = []

    # Faire correspondre les points clés entre les images modèles et la frame
    if des_frame is not None:

        # Correspondance de l'image 1
        matches1 = bf.knnMatch(des1, des_frame, k=2)
        good1 = []
        for m, n in matches1:
            if m.distance < 0.75 * n.distance:
                good1.append([m])
```

```

matched_pts1 = np.float32([kp_frame[m[0].trainIdx].pt for m in good1]).reshape(-1, 2)

# Correspondance de l'image 2
matches2 = bf.knnMatch(des2, des_frame, k=2)
good2 = []
for m, n in matches2:
    if m.distance < 0.75 * n.distance:
        good2.append([m])
matched_pts2 = np.float32([kp_frame[m[0].trainIdx].pt for m in good2]).reshape(-1, 2)

# Correspondance de l'image 3
matches3 = bf.knnMatch(des3, des_frame, k=2)
good3 = []
for m, n in matches3:
    if m.distance < 0.75 * n.distance:
        good3.append([m])
matched_pts3 = np.float32([kp_frame[m[0].trainIdx].pt for m in good3]).reshape(-1, 2)

# Correspondance de l'image 4
matches4 = bf.knnMatch(des4, des_frame, k=2)
good4 = []
for m, n in matches4:
    if m.distance < 0.75 * n.distance:
        good4.append([m])
matched_pts4 = np.float32([kp_frame[m[0].trainIdx].pt for m in good4]).reshape(-1, 2)

# Correspondance de l'image 5
matches5 = bf.knnMatch(des5, des_frame, k=2)
good5 = []
for m, n in matches5:
    if m.distance < 0.75 * n.distance:
        good5.append([m])
matched_pts5 = np.float32([kp_frame[m[0].trainIdx].pt for m in good5]).reshape(-1, 2)

# Ajuster les coordonnées pour le tiers droit de la frame
matched_pts1[:, 0] += (2 * width) // 3
matched_pts2[:, 0] += (2 * width) // 3
matched_pts3[:, 0] += (2 * width) // 3
matched_pts4[:, 0] += (2 * width) // 3
matched_pts5[:, 0] += (2 * width) // 3

# Supprimer les valeurs aberrantes des points clés correspondants pour chaque image modèle
cleaned_coordinates1 = remove_outliers(matched_pts1, threshold=50)
cleaned_coordinates2 = remove_outliers(matched_pts2, threshold=50)
cleaned_coordinates3 = remove_outliers(matched_pts3, threshold=50)
cleaned_coordinates4 = remove_outliers(matched_pts4, threshold=50)
cleaned_coordinates5 = remove_outliers(matched_pts5, threshold=50)

# Déterminer le type d'objet en fonction du nombre de points clés détectés
if len(cleaned_coordinates1) > 0 or len(cleaned_coordinates2) or len(cleaned_coordinates3) or
len(cleaned_coordinates4) or len(cleaned_coordinates5) > 0:
    if len(cleaned_coordinates1) > len(cleaned_coordinates2) and len(cleaned_coordinates1) >
len(cleaned_coordinates3) and len(cleaned_coordinates1) > len(cleaned_coordinates4) and
len(cleaned_coordinates1) > len(cleaned_coordinates5):
        object_type = "STOP"
        cleaned_coordinates = cleaned_coordinates1

```



```

        if len(cleaned_coordinates2) > len(cleaned_coordinates1) and len(cleaned_coordinates2) >
len(cleaned_coordinates3) and len(cleaned_coordinates2) > len(cleaned_coordinates4) and
len(cleaned_coordinates2) > len(cleaned_coordinates5):
            object_type = "30"
            cleaned_coordinates = cleaned_coordinates2
        if len(cleaned_coordinates3) > len(cleaned_coordinates2) and len(cleaned_coordinates3) >
len(cleaned_coordinates1) and len(cleaned_coordinates3) > len(cleaned_coordinates4) and
len(cleaned_coordinates3) > len(cleaned_coordinates5):
            object_type = "50"
            cleaned_coordinates = cleaned_coordinates3
        if len(cleaned_coordinates4) > len(cleaned_coordinates2) and len(cleaned_coordinates4) >
len(cleaned_coordinates3) and len(cleaned_coordinates4) > len(cleaned_coordinates1) and
len(cleaned_coordinates4) > len(cleaned_coordinates5):
            object_type = "PIETON"
            cleaned_coordinates = cleaned_coordinates4
        if len(cleaned_coordinates5) > len(cleaned_coordinates2) and len(cleaned_coordinates5) >
len(cleaned_coordinates3) and len(cleaned_coordinates5) > len(cleaned_coordinates4) and
len(cleaned_coordinates5) > len(cleaned_coordinates1):
            object_type = "RP"
            cleaned_coordinates = cleaned_coordinates5

# Calculer la position moyenne des coordonnées détectées
average_coord = average_coordinates(cleaned_coordinates)

# Vérifier la position la plus proche par rapport à la position précédente
if last_position is not None:
    distances = [distance(average_coord, pos) for pos in cleaned_coordinates]
    closest_index = np.argmin(distances)
    closest_position = cleaned_coordinates[closest_index]
    if distance(closest_position, last_position) < 50:
        average_coord = closest_position

last_position = average_coord

# Extraire les coordonnées moyennes et annoter le type d'objet détecté sur la frame
mean_x = average_coord[0]
mean_y = average_coord[1]
object_size = 80
cv.rectangle(frame, (int(mean_x - object_size / 2), int(mean_y - object_size / 2)),
              (int(mean_x + object_size / 2), int(mean_y + object_size / 2)), (0, 255, 0), 2)
cv.putText(frame, object_type, (int(mean_x) - 20, int(mean_y) - 20),
           cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)

# Afficher les frames annotées
cv.imshow('frame', frame)

# Attendre avant de passer à la frame suivante, arrêter si la touche 'b' est pressée
if cv.waitKey(int(1000/fps)) == ord('b'):
    break

# Libérer les ressources de capture vidéo et fermer les fenêtres
vid.release()
cv.destroyAllWindows()

```


Images obtenues

Détection de Panneaux STOP



Détection de panneaux 30 et ralentisseur





Détection de panneaux 50



Détection de panneaux Passage piéton



Détection de panneaux de Rondpoint



Images de référence pour la détection de panneaux

