CaptureCount

détecteur et compteur d'objets basé sur le Raspberry Pi 5

Saad Imtiaz (Elektor)

La vision artificielle est une technologie fascinante qui permet la reconnaissance et la classification d'une large gamme d'objets de notre environnement. Cet article présente un étrange projet à base de Raspberry Pi qui utilise le modèle de détection d'objet YOLO. Le projet est capable d'identifier de nombreux objets, depuis des choses courantes comme des voitures et des vélos jusqu'aux différents animaux tels que des chats, chiens ou oiseaux.

Figure 1. CaptureCount tournant sur un Raspberry Pi 5 avec le module Caméra du Raspberry Pi (Wide).

Le projet a débuté avec un but précis : créer une détection d'objets omnidirectionnelle avec un système de comptage qui reconnaît le type d'objet et qui compte les objets de chaque catégorie détectée. Le Raspberry Pi était un choix évident par ses fonctionnalités et support dans les projets d'IA. Comme c'était mon premier projet de vision artificielle, il m'a fallu un peu de temps pour apprendre le langage Python et utiliser les bons outils et bibliothèques pour développer ce projet. Comme il y a énormément de projets, de ressources [1][2] et grâce à notre cher ChatGPT, il n'a pas été difficile d'apprendre et de concevoir un tel projet.

Détection d'objet

Dans la recherche de l'élaboration de ce projet, le voyage a commencé par la recherche du bon modèle de détection d'objets, de préférence un modèle qui peut détecter et reconnaître une grande variété d'objets avec un temps d'apprentissage minimal. Mais avant d'aborder cette recherche de modèle de détection d'objets, arrêtons-nous un peu et examinons comment fonctionne la détection d'objets.

La détection d'objets est une technologie qui permet aux ordinateurs d'identifier et localiser des objets dans une image ou une vidéo. Contrairement à la reconnaissance d'image qui vous dit ce qu'il y a dans une image, la détection d'objets va plus loin en pointant exactement l'endroit où ces objets se trouvent et en les surlignant. Cette opération est souvent accomplie au travers de deux processus clés :

- Localisation de l'objet : détermine la localisation d'un objet unique dans une image. Le modèle fournit les coordonnées de la boîte entourant l'objet.
- Classification de l'objet : identifie la nature de l'objet dans la boîte à partir d'une série de catégories connues.

Les modèles avancés de détection d'objets intègrent ces deux étapes en traitant efficacement une image pour fournir à la fois la localisation et la classification de multiples objets dans cette image.

Examen des modèles de détection d'objet

La recherche du modèle idéal de détection d'objets a impliqué l'évaluation de plusieurs options, chacune avec ses avantages et ses limitations. Les modèles tels que R-CNN et ses dérivés offrent une grande précision mais avec une vitesse de traitement lente, incompatible avec des applications en temps réel. Le modèle « Single Shot Multibox Detector « (SSD) présente une alternative plus rapide mais avec, en contre-partie, une perte au niveau précision. Le modèle « Mask R-CNN « possède une détection précise, mais est trop complexe pour notre utilisation. Finalement, le modèle YOLO (*You Only Look Once*, vous ne regardez qu'une seule fois) a été choisi pour son équilibre optimal entre la vitesse, l'efficacité et une précision raisonnable. Sa capacité à traiter les images en temps réel et sa simplicité, alliées à

un bon soutien de la communauté IA, a fait de YOLO le meilleur choix pour l'objectif notre projet.
La sélection du modèle YOLO a été un point crucial dans ce projet. YOLO par Joseph Redmon [3], est reconnu par sa capacité à effectuer une détection en temps réel — une caractéristique cruciale pour tout système de détection. Ce qui m'a séduit dans YOLO était son approche particulière dans la détection d'objets.
Contrairement aux méthodes traditionnelles qui traitent une image en plusieurs étapes, YOLO le fait en une seule passe. Ceci accélère le processus et améliore la capacité du modèle à généraliser à partir de son apprentissage, le rendant plus efficace dans la détection d'objets dans les conditions diverses et imprévisibles du monde réel. Cette fonctionnalité a été fondamentale pour ce projet qui visait à la reconnaissance d'un grand nombre d'objets.

Intégration du logiciel matériel et mise en œuvre

Le projet nécessitait le Raspberry Pi 5, connu pour sa puissance de traitement améliorée et un module caméra compatible. Le procédé d'intégration comprenait la mise en place de l'OS du Raspberry Pi, le branchement de la caméra et l'installation de YOLO. Pour démarrer avec le Raspberry Pi, Il faut d'abord installer l'OS sur sa carte microSD. Ceci se fait simplement en téléchargeant l'imageur Raspberry Pi depuis le site officiel du Raspberry Pi [4]. Puis, il faut lancer l'imageur, sélectionner le bon périphérique, sélectionner le dernier OS Raspberry Pi (64 bit), sélectionner la carte microSD et enfin cliquer simplement sur le bouton *Next*.
En quelques minutes, l'OS Raspberry Pi sera installé sur la carte microSD. Ensuite, insérez la carte SD dans la fente pour carte microSD du Raspberry Pi et mettez-le en marche. Vous aurez aussi besoin d'un moniteur, d'un câble Mini HDMI vers HDMI, d'un clavier et d'une souris pour interagir avec le Raspberry Pi la première fois, mais une fois que le VNC Server ou SSH sera activé, le Raspberry Pi pourra être contrôlé à distance par votre ordinateur par wifi ou par la connexion Ethernet.
Il est temps maintenant d'installer les bibliothèques requises pour ce projet. L'installation des bibliothèques est basique ; cela se fait via le Terminal. Avant d'installer de nouvelles bibliothèques, il est recommandé de mettre à jour les paquets du système en tapant :

```
sudo apt-get update && sudo apt-get upgrade
```

Ceci peut prendre un certain temps et ensuite seulement, nous pouvons installer les bibliothèques nécessaires au projet, en tapant les commandes suivantes :

› Installation des paquets Image I/O

```
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
```

› Installation des paquets Video I/O et les bibliothèques de développement GTK

```
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt-get install libgtk2.0-dev
```

› Dépendances supplémentaires pour OpenCV

```
sudo apt-get install libatlas-base-dev gfortran
```

› Installation de *pip* et *pipx* (outil de gestion des paquets)

```
sudo apt-get install python3-pip
sudo apt install pipx -y
```

› Installation de Numpy, Pandas et Open CV

```
pip install numpy
pip install pandas
sudo apt install python3-opencv
```

Après l'installation de toutes les bibliothèques, il est important de s'assurer d'une communication sans faille entre la partie matérielle et la partie logicielle pour permettre au Raspberry Pi de traiter efficacement les entrées en direct de la caméra. Pour conserver un encombrement minimal, la caméra du Raspberry Pi a été directement branchée sur le port MIPI du Raspberry Pi 5. Maintenant que tout est installé, parlons du code du projet.

Plongeons-nous dans le code

Le cœur de ce projet réside dans son code en Python, qui est disponible sur GitHub [4]. Le script commence par l'importation des bibliothèques essentielles comme OpenCV pour le traitement de l'image, Pandas pour la gestion des données et Numpy pour les opérations numériques.

```
import cv2
import pandas as pd
import numpy as np
import subprocess
import os
from datetime import datetime
```

Au lancement, le script importe les bibliothèques essentielles. *cv2* (OpenCV) est crucial pour les tâches de traitement d'image. *pandas* et *numpy* se chargent respectivement de la manipulation des données et des calculs numériques. *subprocess* et *os* sont des bibliothèques Python standard pour interagir avec le système, comme faire tourner des commandes externes et gérer les chemins d'accès aux fichiers. *datetime* est utilisé pour la détection de l'horodatage.

```
model = './yolo/yolov3.weights'
config = './yolo/yolov3.cfg'
net = cv2.dnn.readNetFromDarknet(config, model)
```

Le script spécifie les chemins vers le fichier de configuration et des valeurs de pré-configuration de YOLO, puis les charge en utilisant le module du réseau neuronal profond OpenCV (DNN). Cette étape initialise le modèle YOLO pour la détection d'objets.

```
classes = []
with open("./yolo/coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
```

Le fichier *coco.names*, contenant le nom des objets que YOLO peut détecter, est lu ligne par ligne pour créer une liste de noms de classe.

Plusieurs fonctions sont définies pour optimiser le processus de détection. Jetons un coup d'œil dans le **listage 1** pour voir les fonctions et les paramètres de départ de la boucle principale.
La fonction `get_output_layers(net)` extrait les noms des couches de sortie du réseau YOLO. L'architecture YOLO a plusieurs couches de sortie et cette fonction aide à les identifier pour traiter les détections.
`draw_bounding_box(…)` dessine les rectangles (*bounding boxes*) autour des objets détectés et les annote avec le nom de l'objet et l'indice de fiabilité.
`calculate_centroid(x, y, w, h)` calcule le point central du rectangle entourant l'objet détecté, point critique pour suivre les objets d'une image à l'autre.

```
def capture_image(image_path):
    subprocess.run(["libcamera-still", "-o", image_path,
"--width", "1920", "--height", "1080", "-n", "-t", "1"],
stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT)
```

Cette fonction utilise le module `subprocess` pour faire tourner *libcamera-still*, un outil de commande en ligne de l'OS du Raspberry Pi, qui capture une image de la caméra et la sauvegarde dans le chemin spécifié. La résolution est 1,920×1,080. Une résolution plus basse peut aussi être utilisée pour minimiser la charge du CPU et également du GPU. Pour y arriver, le fragment du code ci-dessus peut être ajusté comme suit :

```
def capture_image(image_path):
    subprocess.run(["libcamera-still", "-o", image_path,
"--width", "1280", "--height", "720", "-n", "-t", "1"],
stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT)
```

Avant d'entrer dans la boucle principale de détection, le script initialise un `DataFrame` Pandas pour ajouter aux détections, un horodatage et un dictionnaire pour suivre la trace du comptage de chaque type d'objet. `centroid_tracking` est utilisé pour suivre le mouvement des objets.

```
data_frame = pd.DataFrame(columns=['Timestamp', 'Type',
'Count'])
object_counts = {cls: 0 for cls in classes}
centroid_tracking = {}
```

La portion suivante du code (voir **listage 2**) vérifie si l'objet détecté est nouveau ou s'il a déjà été détecté auparavant. Elle utilise `centroid_tracking` pour le déterminer. Si un objet est identifié comme nouveau, elle incrémente le comptage de ce type d'objet, ajoute l'heure de détection et met à jour le DataFrame Pandas. Ce `DataFrame` peut être exporté plus tard au format *.csv* pour une analyse ultérieure.

Configuration de la boucle principale de détection

La boucle principale de détection du Raspberry Pi 5 et du système YOLO est la pièce maîtresse par sa fonctionnalité. Elle débute par la capture d'une image de la caméra du Raspberry Pi, puis par la conversion dans un format conforme au traitement par YOLO. Le système traite ensuite cette image au travers de YOLO, en extrayant les informations vitales telles que l'identification de classe, l'indice de confiance et les coordonnées du rectangle de sélection.
Pour affiner les détections, un filtre « Non-Maximum Suppression « (NMS) est appliqué, éliminant les détections redondantes et moins pertinentes. Le système dessine ensuite des rectangles de sélection autour des objets détectés et, si de nouveaux objets sont détectés, sauvegarde ces images. Tout au long de ce traitement, le système met à jour les structures de données de suivi et d'enregistrement pour maintenir un enregistrement continu des détections. Cette boucle est primordiale dans les capacités de détection en temps réel du Raspberry Pi 5 et du système YOLO. Voir **listage 3** pour la boucle principale du code complet.
Après le traitement des détections, le script compile les données dans un DataFrame et le sauvegarde dans un fichier *.csv*. Ceci permet un examen détaillé de chaque détection.

CaptureCount en action

Lorsqu'il est en action, le programme CaptureCount fait preuve d'une bonne détection des objets, bien qu'accompagné de quelques travers intrigants. Le Raspberry Pi était monté sur un tripode de caméra et la caméra visait au travers d'une fenêtre comme on peut le voir sur la **figure 2**. Comme vous pouvez le constater, je vis dans un quartier urbain de la ville et les seules choses que CaptureCount détecte sont les voitures, les camions, les passants et les motos. Il aurait été préférable d'être à la campagne pour pouvoir capturer un peu de vie sauvage.
L'étonnante performance était caractérisée par un admirable taux de précision de 80 % dans l'identification et la catégorisation des différents objets. Ce niveau de précision était particulièrement remarquable en

Figure 2. Point de vue de la caméra.

Figure 3. Deux voitures détectées par CaptureCount.

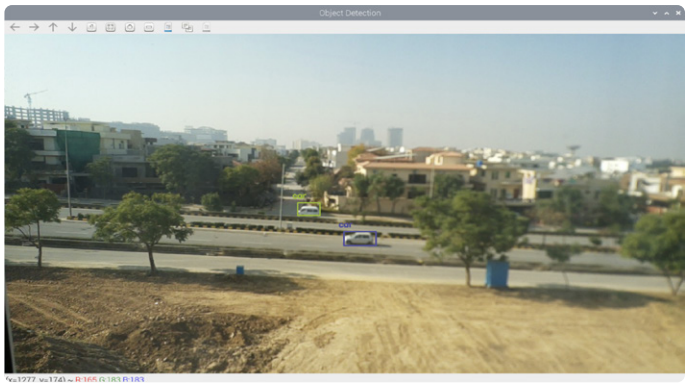


Figure 4. Image des voitures détectées par le système.

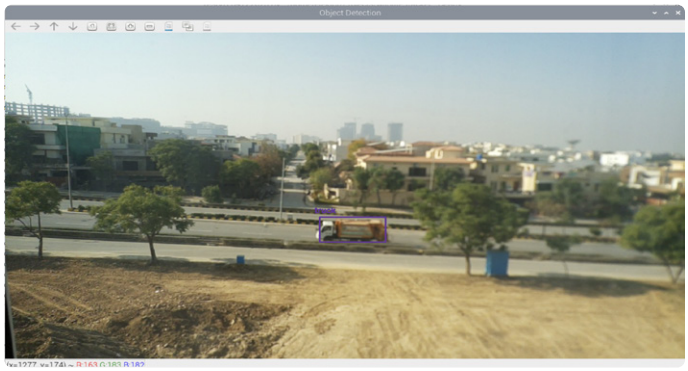


Figure 5. Camion détecté par le système.

considérant l'emplacement de la caméra qui était relativement éloignée des objets, ajouté à l'utilisation d'un objectif grand-angle. Dans les **figure 3**, **4** et **5,** on peut voir les images des objets détectés. Dans les **figure 6** et **figure 7**, le rapport complet est montré dans un fichier *.csv* qui indique quand l'objet a été détecté et combien d'objets de la même catégorie ont été détectés pendant un intervalle de temps donné.

Problèmes rencontrés et observations particulières

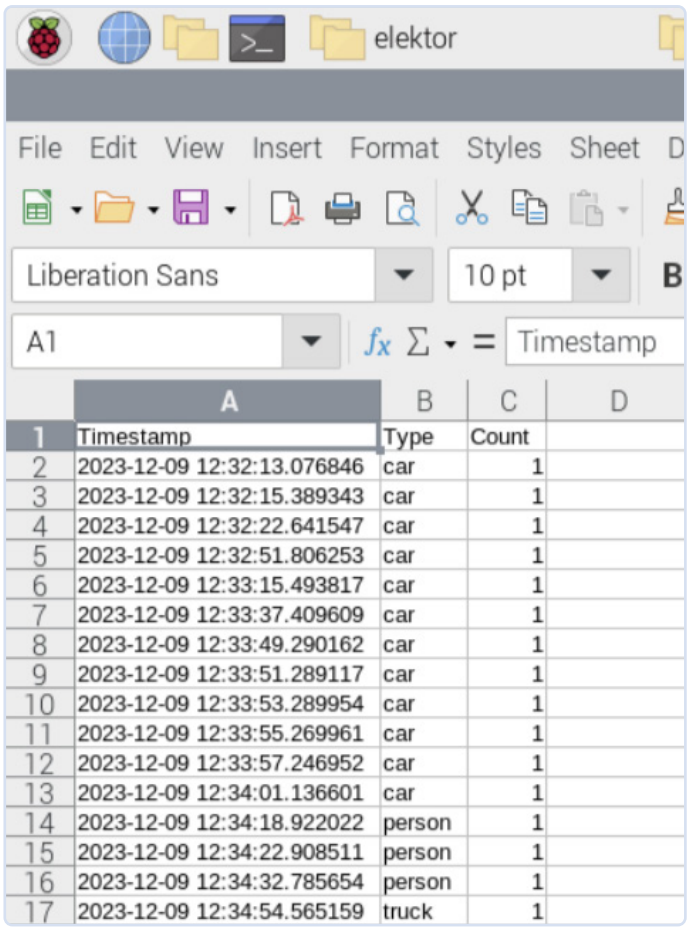
Malgré son succès, le système a rencontré quelques problèmes intéressants :

- Détection sélective dans les réglages dynamiques : certains cas ont été notés où le système a identifié une personne sur une moto mais a omis de détecter la moto elle-même. Cette détection sélective a mis en avant le problème de pouvoir distinguer avec précision des objets étroitement liés, en particulier lorsqu'ils sont en mouvement.
- Mauvaise classification sporadique : il y a eu quelques rares occasions où des piétons ont été pris pour des vélos. Cette mauvaise classification met l'accent sur la complexité impliquée dans la classification des objets, en particulier lorsque les objets partagent des caractéristiques spatiales similaires.

Et le futur de CaptureCount ?

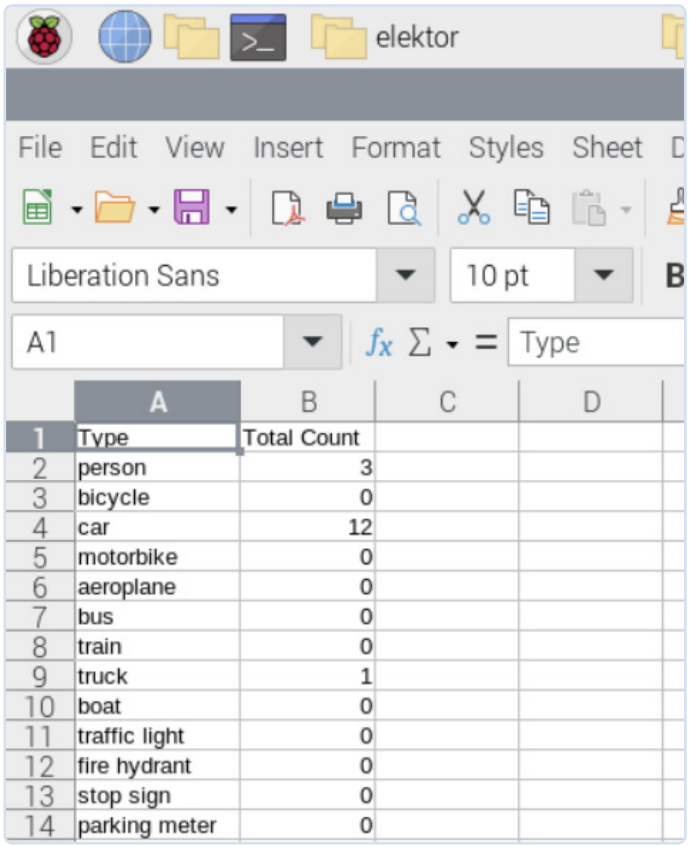
En résumé, ce projet, exploitant le Raspberry Pi 5 et YOLO, a démontré une bonne performance et a ouvert des portes vers diverses applications et améliorations. Les anomalies de détection du système et les mauvaises classifications ponctuelles, non seulement soulignent les domaines pour de futures améliorations, mais aussi ouvrent des opportunités vers plus de recherches.

Des améliorations futures peuvent inclure l'intégration de servomoteurs pour le suivi d'objets déterminés et une connectivité IdO pour des réponses automatiques. Juste un exemple : faire clignoter une



	A	B	C	D
1	Timestamp	Type	Count	
2	2023-12-09 12:32:13.076846	car	1	
3	2023-12-09 12:32:15.389343	car	1	
4	2023-12-09 12:32:22.641547	car	1	
5	2023-12-09 12:32:51.806253	car	1	
6	2023-12-09 12:33:15.493817	car	1	
7	2023-12-09 12:33:37.409609	car	1	
8	2023-12-09 12:33:49.290162	car	1	
9	2023-12-09 12:33:51.289117	car	1	
10	2023-12-09 12:33:53.289954	car	1	
11	2023-12-09 12:33:55.269961	car	1	
12	2023-12-09 12:33:57.246952	car	1	
13	2023-12-09 12:34:01.136601	car	1	
14	2023-12-09 12:34:18.922022	person	1	
15	2023-12-09 12:34:22.908511	person	1	
16	2023-12-09 12:34:32.785654	person	1	
17	2023-12-09 12:34:54.565159	truck	1	

Figure 6. Capture d'écran du fichier .csv montrant les objets détectés avec respectivement leurs horodatages.



	A	B	C	D
1	Type	Total Count		
2	person	3		
3	bicycle	0		
4	car	12		
5	motorbike	0		
6	aeroplane	0		
7	bus	0		
8	train	0		
9	truck	1		
10	boat	0		
11	traffic light	0		
12	fire hydrant	0		
13	stop sign	0		
14	parking meter	0		

Figure 7. Capture d'écran du fichier .csv montrant la somme totale des objets détectés dans une catégorie donnée.

LED RGB pour un type spécifique d'objet. Pour vous donner quelques astuces pour enrichir votre projet avec cette idée ou tout autre contrôle de matériel, voir **listage 4.** Cette fonction allume la LED rouge pour une voiture, verte pour une personne et rouge et vert (donnant du jaune) pour un camion. Après un délai, toutes les LED sont éteintes. De plus, des modifications peuvent être effectuées pour des applications plus vastes dans la surveillance avancée, la gestion des foules et du trafic, la surveillance de la faune, les statistiques de vente et la santé. La capacité en temps réel du système et la détection d'objets précis posent les fondations de solutions innovantes pour diverses industries en montrant les possibilités étendues dans les domaines de l'IA et de la vision par ordinateur.

VF : Chris Elsass — 230749-04

Questions ou commentaires ?

Envoyez un courriel à l'auteur (saad.imtiaz@elektor.com) ou contactez Elektor (redaction@elektor.fr).

À propos de l'Auteur

Saad Imtiaz (Ingénieur Expérimenté, Elektor) est un ingénieur en mécatronique avec une expérience dans les systèmes embarqués, les systèmes mécatroniques et le développement de produits. Il a collaboré avec de nombreuses entreprises, allant des startup jusqu'aux entreprises mondiales, dans le développement et le prototypage. Saad a aussi passé du temps dans l'industrie aéronautique et a dirigé une startup technologique. Chez Elektor, il dirige des développement de projets dans les domaines matériel et logiciel.

Produits

- **Raspberry Pi 5 Ultimate Starter Kit (8 GB)**
  www.elektor.fr/20721
- **Raspberry Pi High Quality Camera Module**
  www.elektor.fr/19279
- **Raspberry Pi Camera Module 3 Wide**
  www.elektor.fr/20364

LIENS

[1] Implementation of YOLOv3: Simplified: https://analyticsvidhya.com/blog/2021/06/implementation-of-yolov3-simplified
[2] YOLOv3 code explained: https://pylessons.com/YOLOv3-code-explanation
[3] YOLO: Real-Time Object Detection: https://pjreddie.com/darknet/yolo
[4] Dépôt Github de CaptureCount : https://github.com/ElektorLabs/CaptureCount

Listage 1. Fonctions de détection

```
import cv2
import pandas as pd
import numpy as np
import subprocess
import os
from datetime import datetime

# Load pre-trained model and configuration
model = './yolo/yolov3.weights'  # Update this path to your model's path
config = './yolo/yolov3.cfg'     # Update this path to your config file's path
net = cv2.dnn.readNetFromDarknet(config, model)

# Load class names
classes = []
with open("./yolo/coco.names", "r") as f:  # Update to the path of your coco.names file
    classes = [line.strip() for line in f.readlines()]

# Function to get output layers
def get_output_layers(net):
    layer_names = net.getLayerNames()
    return [layer_names[i - 1] for i in net.getUnconnectedOutLayers().flatten()]
```

```
# Function to draw bounding box
def draw_bounding_box(img, class_id, confidence, x, y, x_plus_w, y_plus_h):
    label = str(classes[class_id])
    color = np.random.uniform(0, 255, size=(3,))
    cv2.rectangle(img, (x, y), (x_plus_w, y_plus_h), color, 2)
    cv2.putText(img, label, (x-10, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

# Function to calculate centroid of a bounding box
def calculate_centroid(x, y, w, h):
    return (int(x + w/2), int(y + h/2))

# Function to capture image using libcamera-still
def capture_image(image_path):
    subprocess.run(["libcamera-still", "-o", image_path, "--width", "1920", "--height", "1080", "-n", "-t",
                                                "1"], stdout=subprocess.DEVNULL, stderr=subprocess.STDOUT)

# Initialize a DataFrame to store counts
data_frame = pd.DataFrame(columns=['Timestamp', 'Type', 'Count'])
object_counts =  # Object count per category
centroid_tracking = {}  # Tracks centroids of detected objects

# Ensure output directory exists
output_dir = "output"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
```

Listage 2. Suivi du point central — détecte si un objet est nouveau ou s'il a déjà été détecté

```
# Check if this object was already detected
if class_id not in centroid_tracking or not any(np.linalg.norm(np.array(centroid) - np.array(old_centroid))
                                                < 50 for old_centroid in centroid_tracking[class_id]):
    object_counts[classes[class_id]] += 1
    centroid_tracking.setdefault(class_id, []).append(centroid)
    new_row = pd.DataFrame([{'Timestamp': datetime.now(), 'Type': classes[class_id], 'Count': 1}])
    data_frame = pd.concat([data_frame, new_row], ignore_index=True)
```

Listage 3. Boucle main

```
# Main loop
image_path = "temp.jpg"
object_id = 0  # Unique identifier for each object

try:
    while True:
        capture_image(image_path)
        frame = cv2.imread(image_path)
        if frame is None:
            continue

        Width = frame.shape[1]
        Height = frame.shape[0]
        scale = 0.00392
```

(Suite à la page suivante)

```
        # Create a blob and pass it through the model
        blob = cv2.dnn.blobFromImage(frame, scale, (416, 416), (0, 0, 0), True, crop=False)
        net.setInput(blob)
        outs = net.forward(get_output_layers(net))

        # Process the outputs
        class_ids = []
        confidences = []
        boxes = []
        centroids = []
        conf_threshold = 0.5
        nms_threshold = 0.4

        for out in outs:
            for detection in out:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
                if confidence > conf_threshold:
                    center_x = int(detection[0] * Width)
                    center_y = int(detection[1] * Height)
                    w = int(detection[2] * Width)
                    h = int(detection[3] * Height)
                    x = center_x - w / 2
                    y = center_y - h / 2
                    class_ids.append(class_id)
                    confidences.append(float(confidence))
                    boxes.append([x, y, w, h])
                    centroids.append(calculate_centroid(x, y, w, h))

        indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

        for i in indices:
            box = boxes[i]
            x, y, w, h = box
            x, y, w, h = int(x), int(y), int(w), int(h)  # Ensure the values are integers

            centroid = centroids[i]
            class_id = class_ids[i]

            # Check if this object was already detected
            if class_id not in centroid_tracking or not any(np.linalg.norm(np.array(centroid) -
                            np.array(old_centroid)) < 50 for old_centroid in centroid_tracking[class_id]):
                object_counts[classes[class_id]] += 1
                centroid_tracking.setdefault(class_id, []).append(centroid)


                # Update data_frame using pandas.concat
                new_row = pd.DataFrame([{'Timestamp': datetime.now(),'Type': classes[class_id],'Count': 1}])
                data_frame = pd.concat([data_frame, new_row], ignore_index=True)

                # Save the entire frame when an object is detected
                frame_filename = os.path.join(output_dir, f"frame__.jpg")
                cv2.imwrite(frame_filename, frame)
                object_id += 1

            draw_bounding_box(frame, class_id, confidences[i], round(x), round(y), round(x+w), round(y+h))

        # Display the frame
        cv2.imshow("Object Detection", frame)

        # Break loop with 'q' key
```

```
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
finally:
    cv2.destroyAllWindows()
    if os.path.exists(image_path):
        os.remove(image_path)

# Print and save the total count of objects detected per category
total_counts = pd.DataFrame(object_counts.items(), columns=['Type', 'Total Count'])
print(total_counts)
total_counts.to_csv("total_object_counts.csv", index=False)

# Save detailed counts to CSV
data_frame.to_csv("object_counts.csv", index=False)

# Write the total count to a text log
with open("total_counts_log.txt", "w") as log_file:
    log_file.write(str(total_counts))
```

Listage4. Clignotement d'une LED RGB pour les objets détectés

```
#include these libraries in the shared code
import RPi.GPIO as GPIO
import time

# GPIO pin setup
RED_PIN, GREEN_PIN, BLUE_PIN = 17, 27, 22

# Update with your GPIO pin numbers

# add this part in the initial part of the code
GPIO.setmode(GPIO.BCM)
GPIO.setup([RED_PIN, GREEN_PIN, BLUE_PIN], GPIO.OUT)

def blink_rgb_led(object_type):
    GPIO.output(RED_PIN, object_type == 'car' or object_type == 'truck')
    GPIO.output(GREEN_PIN, object_type == 'person' or object_type == 'truck')
    GPIO.output(BLUE_PIN, False)
    time.sleep(1)
    GPIO.output([RED_PIN, GREEN_PIN, BLUE_PIN], False)

# Usage of this function:

# ... [Previous code] ...
for out in outs:
    for detection in out:
# ... [add the following lines to the code to this 'for' loop]
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > conf_threshold:
            detected_object = classes[class_id]
             blink_rgb_led(detected_object) # Blink the LED based on the detected object

# ... [Rest of your detection and saving logic] ...
# ... [Remaining code] ...
```