Systèmes embarqués automobiles

Troisième année du cycle d'ingénieur en génie mécatronique d'automobile

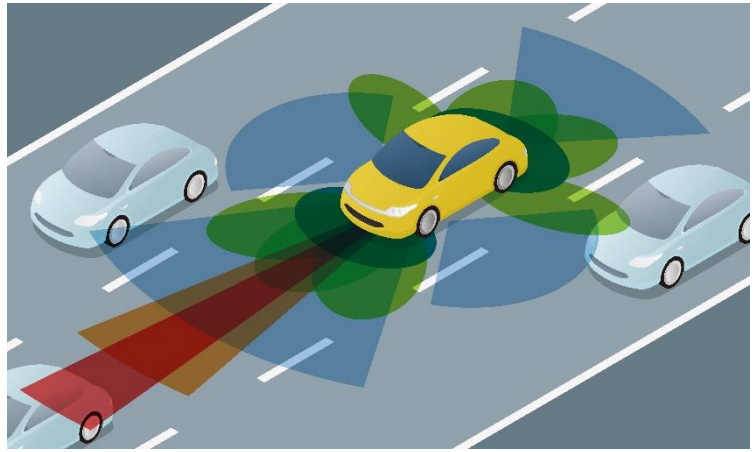
Chapitre 5: ADAS

Année universitaire 2024/2025

Dr. Ing. LASSIOUI Abdellah

ADAS: Advanced Driver Assistance System

1. Les systèmes avancés d'aide à la conduite (ADAS) sont des systèmes électroniques embarqués dans un véhicule qui utilisent des technologies avancées pour assister le conducteur. Ils peuvent inclure de nombreuses fonctions de sécurité active, et les termes « ADAS » et « sécurité active » sont souvent utilisés de manière interchangeable.



2. L'ADAS utilise des capteurs dans le véhicule tels que des radars et des caméras pour percevoir le monde qui l'entoure, puis fournit des informations au conducteur ou prend des mesures automatiques en fonction de ce qu'il perçoit.

ADAS en tant que système d'avertissement

Vs

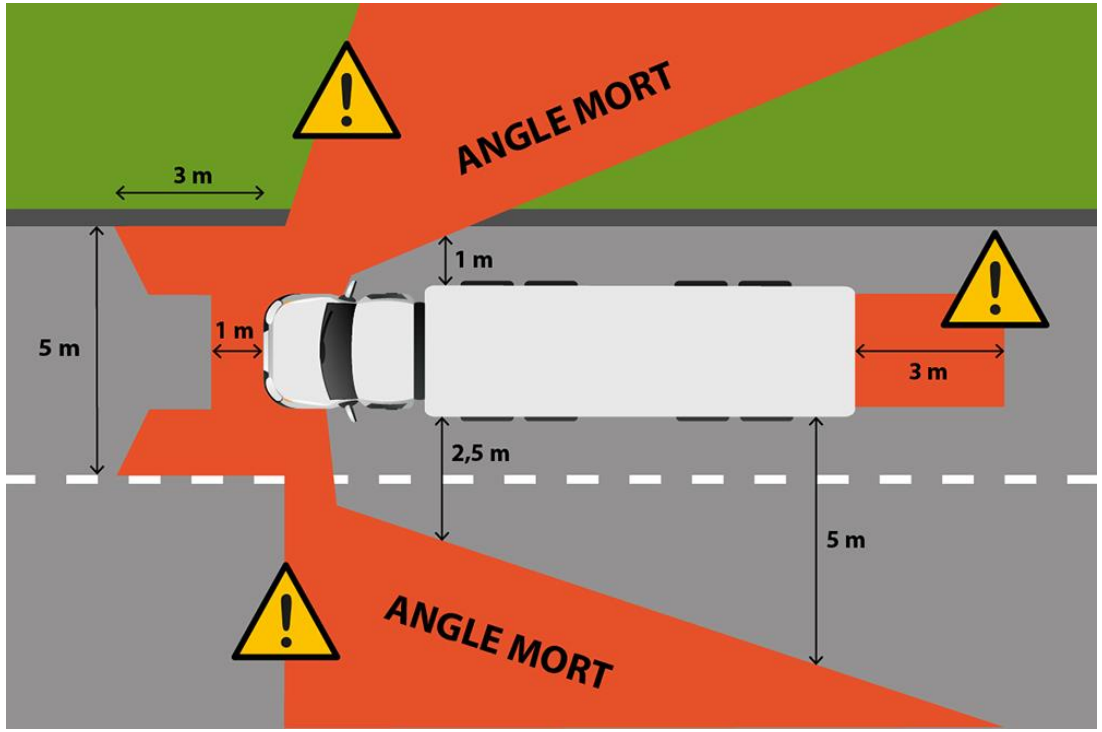
ADAS en tant que système de sécurité active

ADAS en tant que système d'avertissement

Les fonctions ADAS qui fournissent des informations incluent généralement le mot « avertissement » dans leur nom.

Par exemple:

- Si le véhicule détecte un objet tel qu'un autre véhicule ou un cycliste dans un endroit où le conducteur ne peut pas les voir, des fonctions telles que l'avertissement d'angle mort ou l'avertissement de recul arrière alerteront le conducteur.
- De même, si le système détermine que le véhicule s'écarte de sa voie, il peut activer l'avertissement de sortie de voie pour alerter le conducteur.



Source:
Nouvelle norme : adhésifs angles morts - TMS

ADAS en tant que système d'avertissement

Par exemple:

- Système de détection d'angle mort

ADAS en tant que système de sécurité active

Lorsque les détections sont couplées à une technologie qui va au-delà d'un simple avertissement, l'ADAS devient un système de sécurité active, ce qui signifie que le véhicule contrôlera « activement » le freinage ou la direction. Ces fonctionnalités incluent le plus souvent le mot « assistance » dans leur nom.

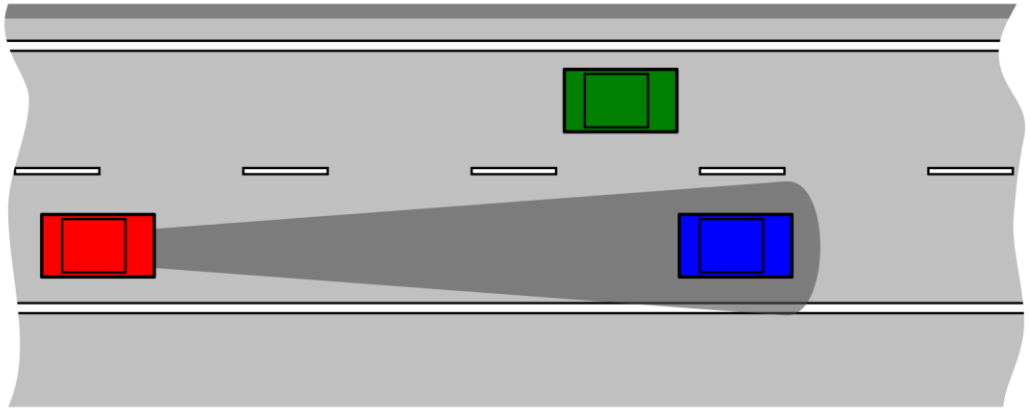
Par exemple:

- Les ADAS comprennent des fonctions de propulsion telles que le régulateur de vitesse adaptatif, qui fait varier la vitesse pour garantir qu'un véhicule maintient une distance de sécurité avec le véhicule qui le précède.
- Des fonctionnalités ADAS plus sophistiquées peuvent même gérer la direction et la propulsion sans nécessiter de contrôle manuel de la part du conducteur dans certaines conditions, comme la conduite sur autoroute ou la circulation en accordéon.

ADAS en tant que système de sécurité active

Exemple : ACC: Adaptive Cruise Control

Le contrôle est basé sur les informations fournies par les capteurs embarqués. Ces systèmes peuvent utiliser un radar, un capteur laser ou une caméra permettant au véhicule de freiner lorsqu'il détecte que la voiture s'approche d'un autre véhicule qui précède, puis d'accélérer lorsque la circulation le permet.



ADAS: les 6 niveaux

- Niveau 2: le véhicule inclue d'autres fonctionnalités: le conducteur supervise le fonctionnement tout le temps: Système de détection d'angle morts, système de freinage d'urgence, Système de régulation de vitesse adaptatif…

- Niveau 3: le véhicule inclue certaines fonctionnalités avancées: reconnaissance des panneaux de signalisation, Définition du chemin libre, positionnement précis, stationnement automatique…
- Mais le conducteur doit prendre le relais dans certaines situations

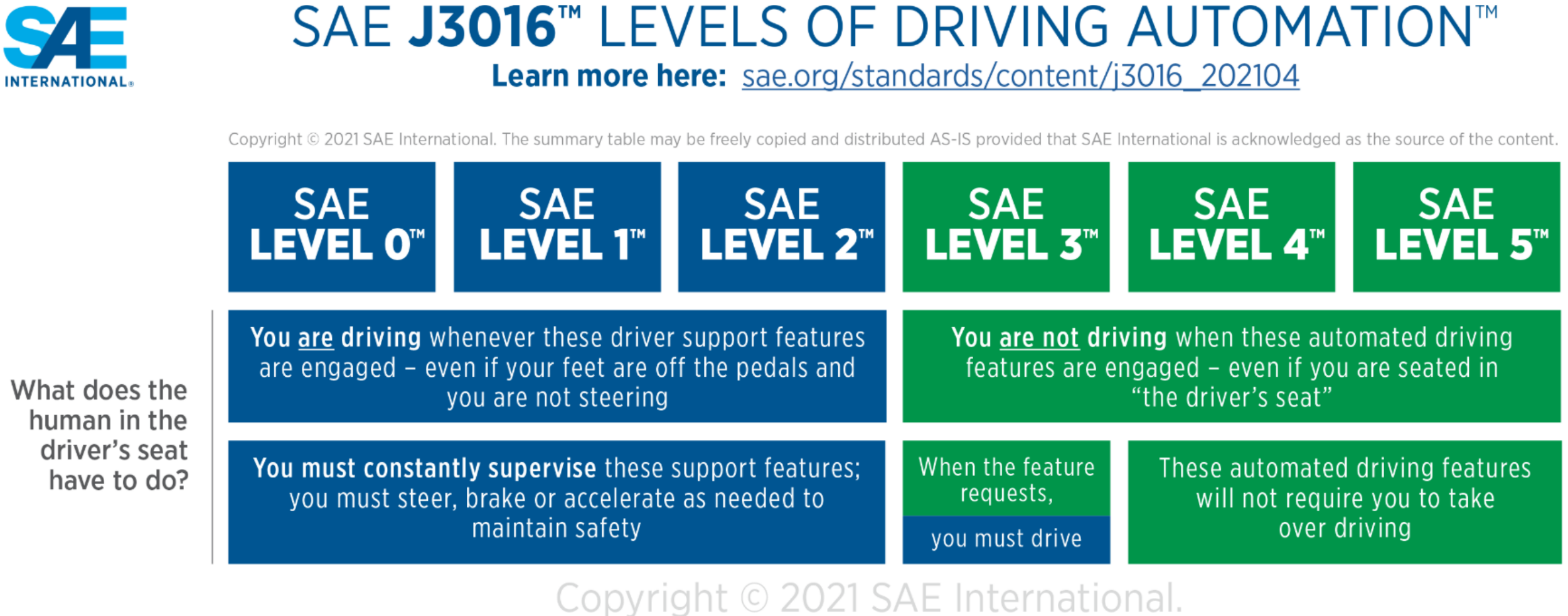
- Niveau 1: DAS :Driver Assistance System: le véhicule inclue certaines fonctionnalités permettant d'assister le conducteur: Cruise Control, ESP, Parking Sensors…

- Niveau 4: le véhicule inclue plusieurs fonctionnalités permettant d'assurer une conduite autonome.

- Niveau 0: le conducteur contrôle toutes les fonctionnalités du véhicule

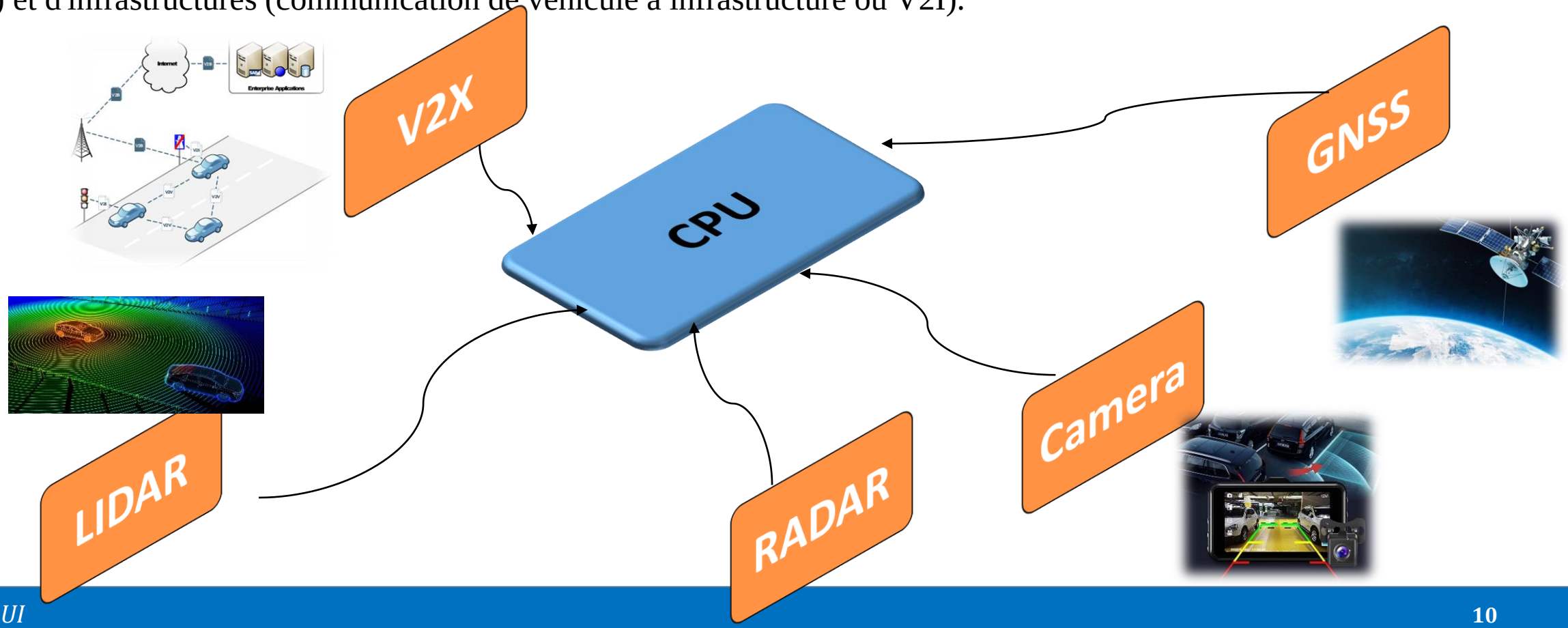
- Niveau 5: le véhicule inclue toutes les fonctionnalités permettant d'assurer une conduite autonome.
- Aucune intervention n'est nécessaire de la part du conducteur

ADAS: les 6 niveaux



ADAS: la notion du sensor fusion

Les systèmes ADAS s'appuient sur des données provenant de plusieurs sources, notamment l'imagerie automobile, le LiDAR, le radar, le traitement d'images, les données de navigation…. Des données supplémentaires sont possibles à partir d'autres sources distinctes de la plate-forme principale du véhicule, notamment d'autres véhicules (communication de véhicule à véhicule ou V2V) et d'infrastructures (communication de véhicule à infrastructure ou V2I).



ADAS: la notion du Sensor Fusion

Définition:

La fusion de capteurs est un processus de combinaison de données de capteurs ou de données dérivées de sources disparates afin que les informations résultantes présentent moins d'incertitude que ce qui serait possible si ces sources étaient utilisées individuellement.

Caméra	
Avantages	**Inconvénients**
Capteur semblable à une vision. Tout comme notre vision, les caméras peuvent facilement distinguer les formes, les couleurs et identifier rapidement le type d'objet en fonction de ces informations. Par conséquent, les caméras peuvent produire une expérience de conduite autonome très similaire à celle produite par un conducteur humain.	Mauvaise vision en cas d'événements météorologiques extrêmes. Sa similitude avec l'œil humain en fait également un inconvénient majeur dans des conditions météorologiques extrêmes telles que des tempêtes de neige, des tempêtes de sable ou d'autres conditions entraînant une faible visibilité.

ADAS: la notion du Sensor Fusion

RADAR

La technologie radar peut être décomposée en un émetteur et un récepteur. L'émetteur envoie des ondes radio dans une direction ciblée. Ces ondes radio sont ensuite réfléchies lorsqu'elles atteignent un objet significatif. Le récepteur capte ces ondes réfléchies et les analyse pour identifier l'emplacement, la vitesse et la direction de l'objet.

RADAR	
Avantages	**Inconvénients**
Insensible aux conditions météorologiques. Le principal avantage du radar est que la transmission des ondes radio n'est pas affectée par la visibilité, l'éclairage et le bruit. Par conséquent, les performances du radar sont constantes dans toutes les conditions environnementales.	Modélisation basse définition. Les ondes radio sont très précises pour détecter les objets. Pourtant, comparé à la caméra, le radar est relativement faible pour modéliser une forme parfaitement précise de l'objet. Par conséquent, le système peut ne pas être en mesure d'identifier exactement de quel objet il s'agit. Par exemple, contrairement à la caméra, le système radar ne peut normalement pas distinguer les vélos des motos, même s'il n'a aucun problème à déterminer leur vitesse.

ADAS: la notion du Sensor Fusion

LIDAR

Le LiDAR (Light Detection and Ranging) est une technologie de télédétection qui utilise des impulsions laser pour mesurer des distances et générer des représentations tridimensionnelles précises d'un environnement

LIDAR	
Avantages	**Inconvénients**
Les LiDAR offrent une très bonne résolution en mesurant précisément la distance à l'aide d'impulsions laser. Cela permet de créer des cartes 3D détaillées et fiables. **Indépendance vis-à-vis de la luminosité ambiante :** Contrairement aux caméras, les LiDAR fonctionnent efficacement dans des conditions de faible éclairage ou en présence de contre-jour, assurant ainsi une détection fiable dans diverses conditions.	**Coût élevé :** Les systèmes LiDAR restent généralement onéreux, ce qui peut limiter leur intégration dans les systèmes ADAS **Complexité d'intégration et de traitement des données :** Les données générées par les LiDAR nécessitent des algorithmes de traitement et de fusion sophistiqués pour être exploitées efficacement, augmentant ainsi la complexité du système.

ADAS: la notion du Sensor Fusion

Les trois types de capteurs ont leurs avantages et leurs inconvénients. Par conséquent, la plupart des constructeurs utilisent un mélange d'au moins deux des trois pour se compléter et compenser leurs faiblesses. À mesure que les technologies de capteurs deviennent plus matures, de plus en plus de véhicules devraient atteindre les niveaux de conduite autonome 3 à 4 dans les cinq prochaines années.

	Caméra	RADAR	LIDAR	Caméra + RADAR + LIDAR
Détection d'objet				
Classification d'objet				
Estimation de la distance				
Visibilité				
Insensibilité aux conditions météo et aux obscurités				
Suivie de la voie				

ADAS:

Traitement distribué avec fusion au niveau des objets

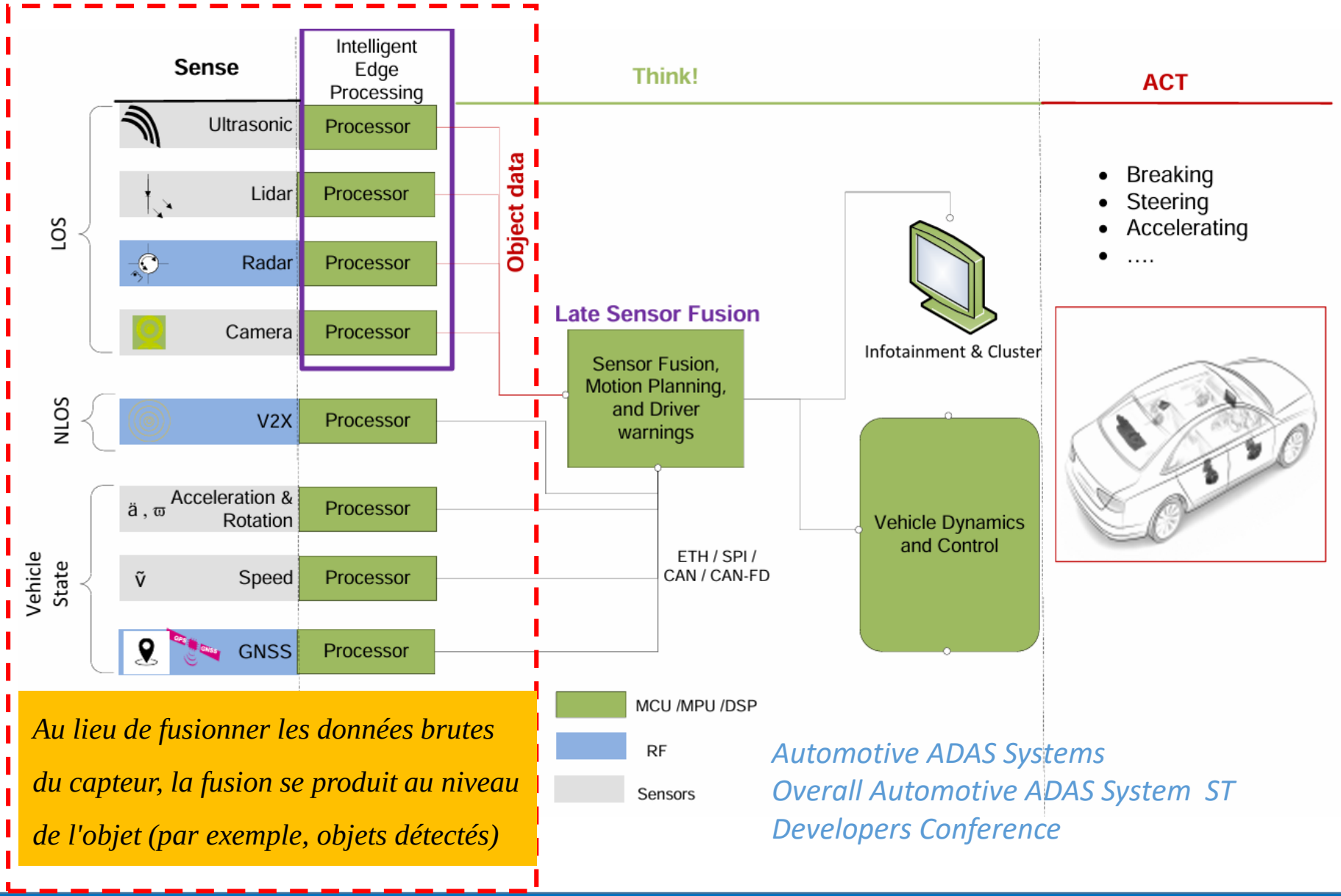
Vs

Traitement distribué avec fusion au niveau des données

ADAS: Traitement distribué avec fusion au niveau des objets

Le traitement distribué avec fusion au niveau des objets fait référence à un système dans lequel plusieurs nœuds de traitement collaborent pour analyser et intégrer des données au niveau des objets provenant de différentes sources afin d'améliorer la prise de décision.

Les données issues des différents nœuds sont fusionnées dans un bus (CAN, SPI, I2C, Ethernet…)



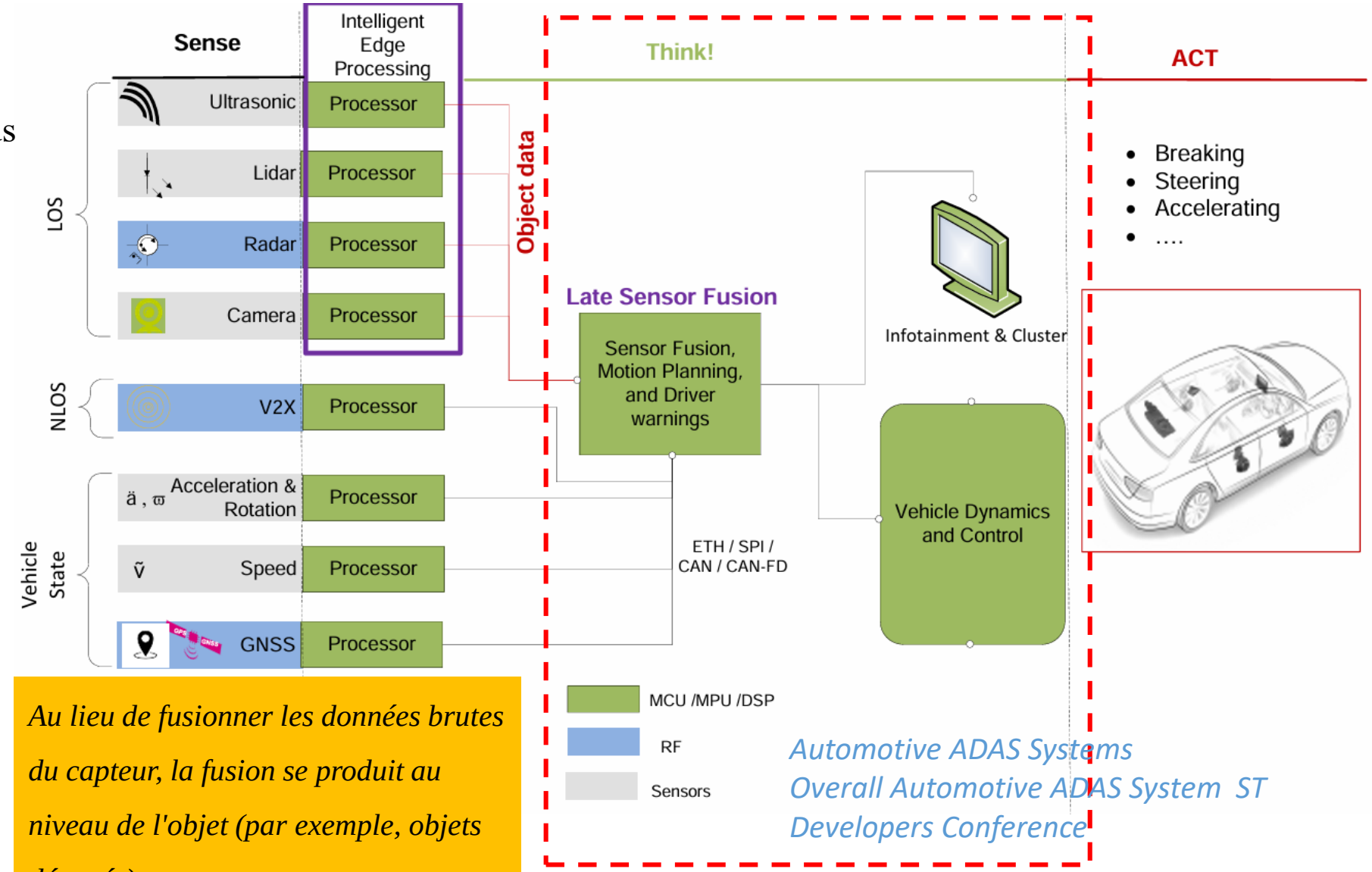
ADAS: Traitement distribué avec fusion au niveau des objets

Les données issues de chaque nœuds sont ensuite communiquées à un nœud centrale pour analyser les données et prendre des décisions.

Exemple:

- Lidar : l'objet est un passager (60%)
- Caméra : L'objet est un passager (80%)
- Radar: un objet détecté avec une probabilité de 100%

La CPU prend la décision finale



Au lieu de fusionner les données brutes du capteur, la fusion se produit au niveau de l'objet (par exemple, objets détectés)

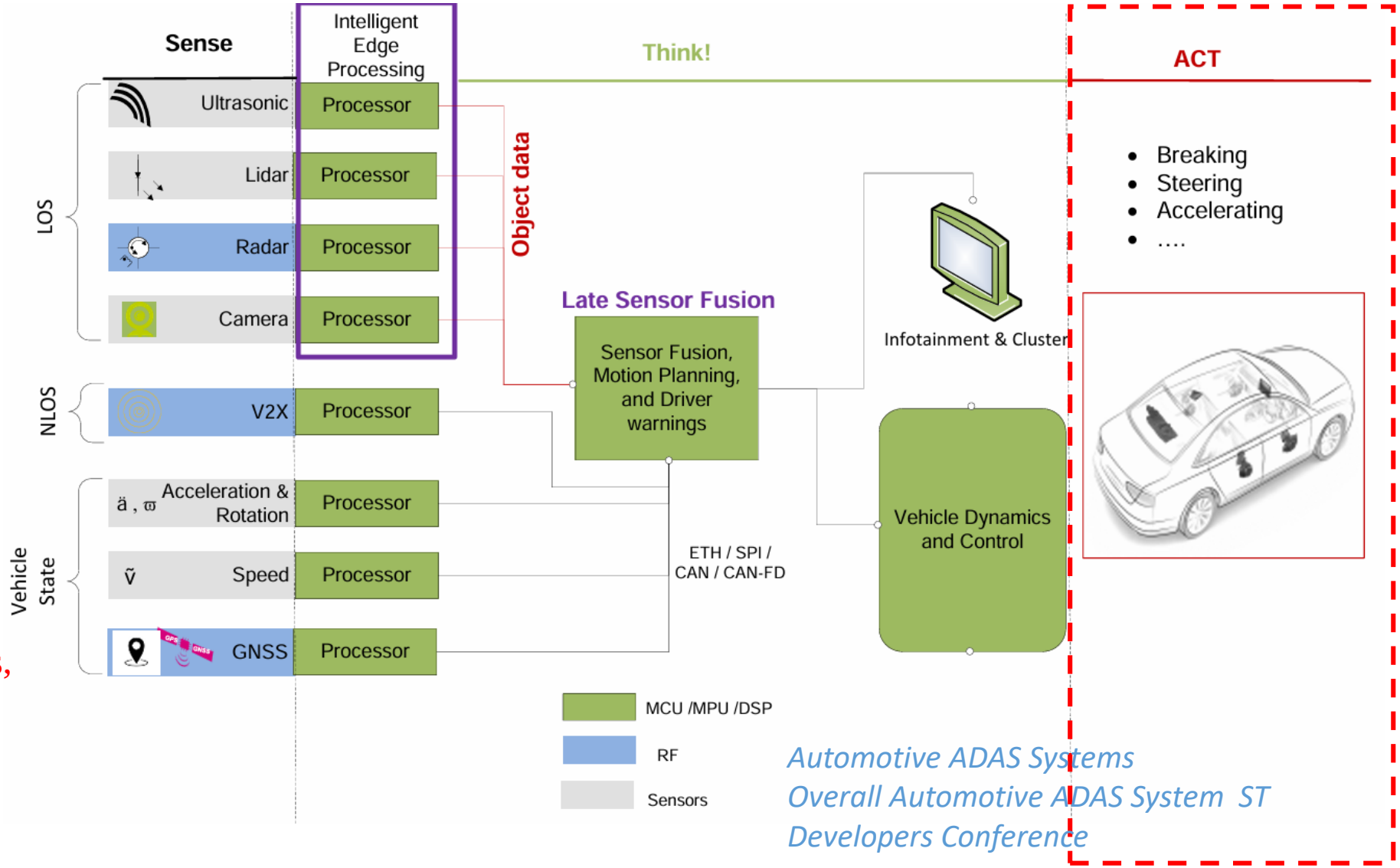
ADAS: Traitement distribué avec fusion au niveau des objets

Prise de décision et résultats

La liste finale des objets fusionnés (position, type, vitesse, confiance) est générée.

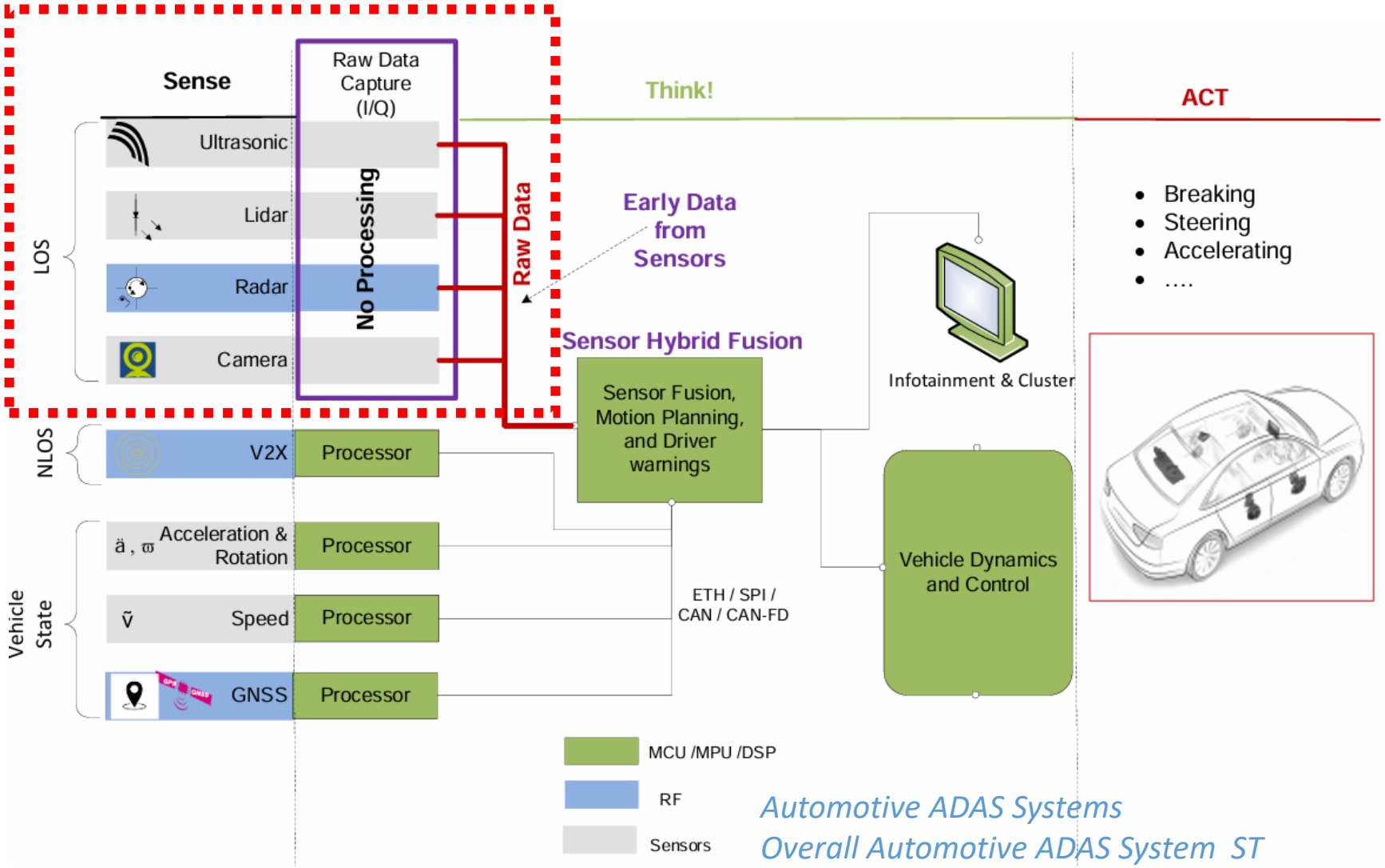
Le processeur utilise ces résultats pour :

Le contrôle autonome du véhicule (par exemple, éviter une collision, ajuster la vitesse, éviter les obstacles, freinage d'urgence…)



ADAS: Traitement centralisé avec fusion au niveau des données

Le traitement centralisé avec fusion de données brutes fait référence à un système dans lequel plusieurs capteurs envoient leurs données brutes non traitées à un processeur central, qui effectue tout le traitement, la fusion des données et la prise de décision.



Automotive ADAS Systems
Overall Automotive ADAS System ST Developers Conference

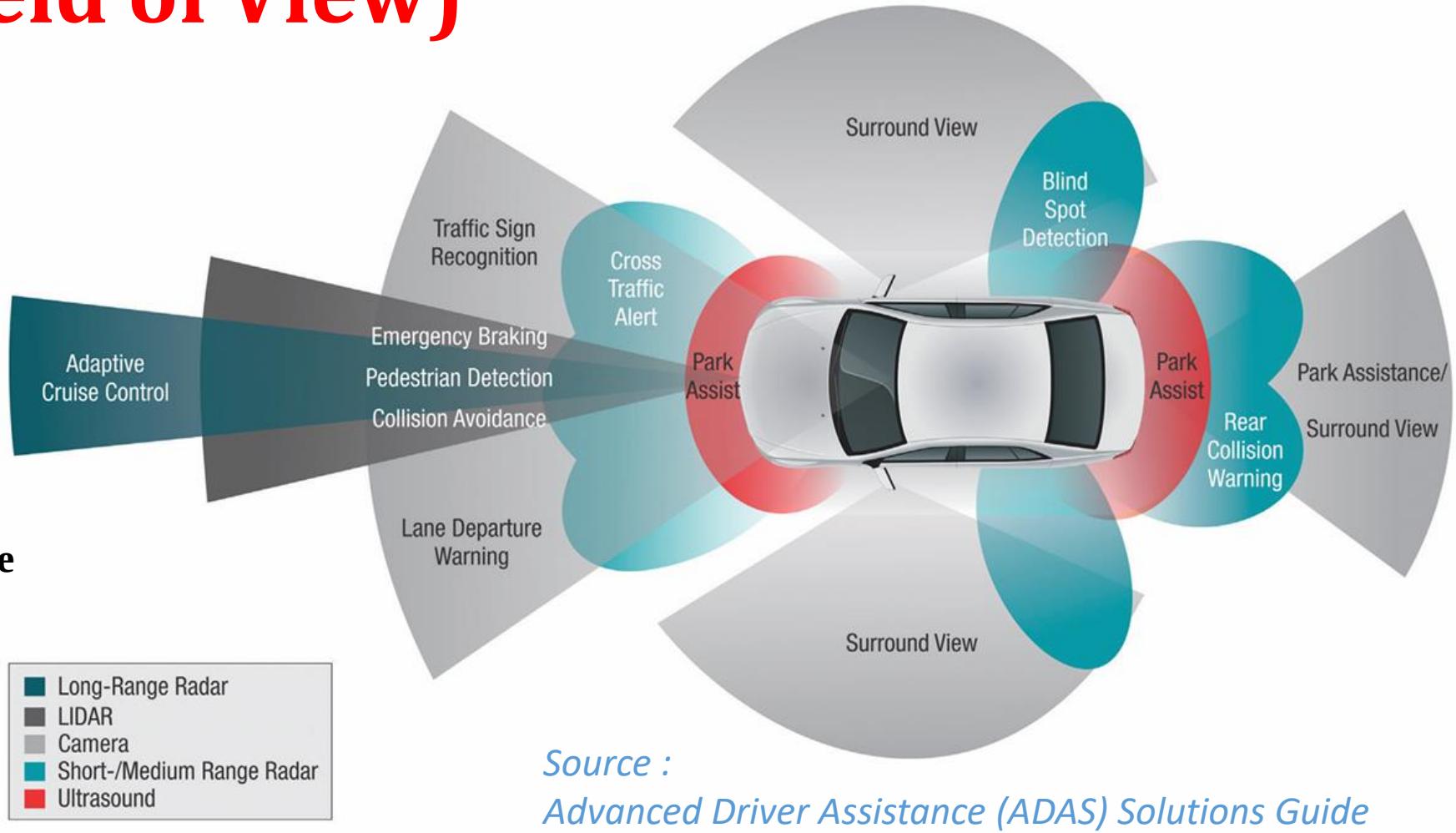
ADAS: Traitement centralisé vs distribué

Critère	Traitement Centralisé (Fusion des Données Brutes)	Traitement Distribué (Fusion des Objets)
Principe	Toutes les données brutes sont envoyées vers un processeur central.	Chaque capteur traite localement avant d'envoyer des données filtrées.
Charge de calcul	Très élevée sur l'unité centrale.	Répartie entre plusieurs unités.
Précision	Très élevée (aucune perte de données avant fusion).	Bonne, mais dépend de la qualité du prétraitement local.
Bande passante	Très élevée (transmission de toutes les données brutes).	Plus faible (transmission des résultats traités).
Latence	Plus élevée (temps de traitement plus long).	Plus faible (traitement parallèle et distribué).
Flexibilité	Facile à modifier au niveau du traitement.	Plus évolutif et modulaire.
Tolérance aux pannes	Faible (dépend d'un processeur central unique).	Élevée (les unités locales peuvent fonctionner indépendamment).

ADAS: les différents capteurs et leurs champ de vision (FoV: Field of View)

La figure suivante illustre le champ de vision (FoV - Field of View) des différents capteurs dans un système ADAS.

Chaque capteur sert à une tâche précise en fonction de son **angles de couverture** et de sa **portée**



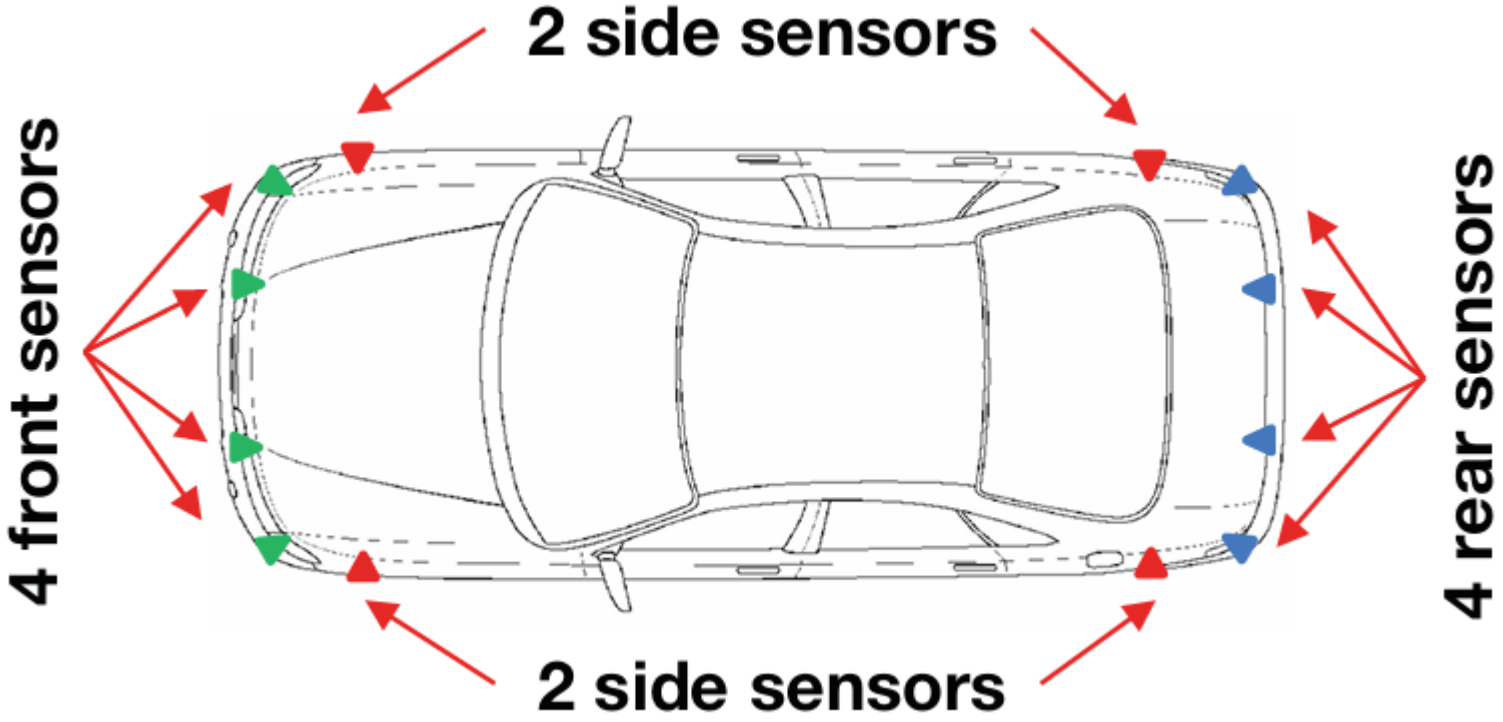
Source :
Advanced Driver Assistance (ADAS) Solutions Guide (Texas Instrument)

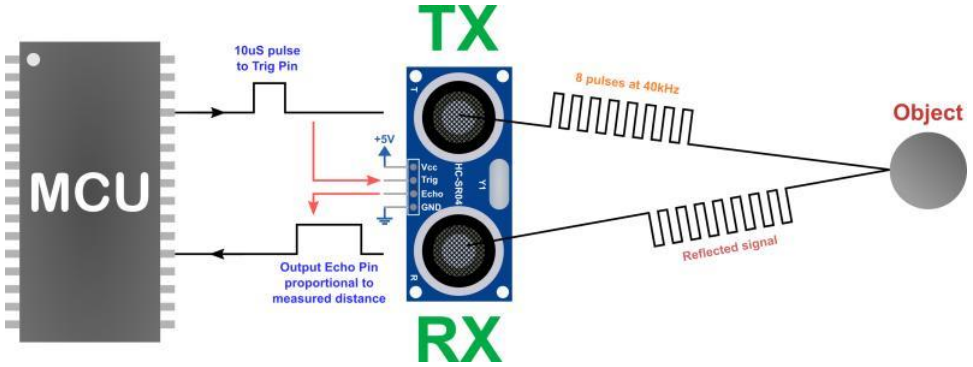
ADAS: exemple d'emplacement des capteurs RADAR, ou ultrason dans un véhicule

Les capteurs à ultrasons sont principalement utilisés dans les applications d'aide au stationnement.

En général, une voiture est équipée de huit à douze de ces capteurs.

Ces capteurs ont une porté de 10cm à 6m en générale

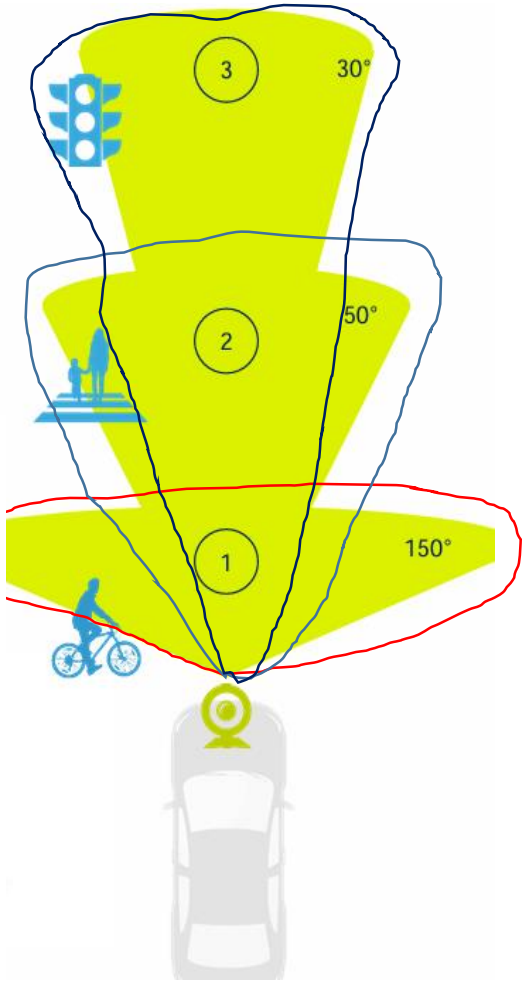




Source :
Advanced Driver Assistance (ADAS) Solutions Guide (Texas Instrument)

ADAS: exemple d'emplacement des caméras

Une **caméra trifocale** est un ensemble de **trois caméras** ayant des objectifs avec des focales différentes pour couvrir simultanément **plusieurs champs de vision**. Ce type de caméra est souvent utilisé dans les systèmes ADAS et les véhicules autonomes pour maximiser la perception de l'environnement.



Objectif	Champ de Vision (FoV)	Portée Approx.	Utilisation
Grand-angle	🌎 120° - 150°	0 – 60 m	Détection des piétons, véhicules proches, panneaux de signalisation.
Standard	🔍 50° - 70°	20 – 150 m	Reconnaissance des marquages au sol, suivi de véhicules.
Téléobjectif	🎯 20° - 30°	50 – 250 m	Détection des objets éloignés (véhicules sur autoroute, anticiper les dangers).

Application: détection d'objet en utilisant YOLOv3 et OpenCV

Application: détection d'objet en utilisant YOLOv3 et OpenCV

YOLO (You Only Look Once) :
Détection d'objets en temps réel
YOLO (You Only Look Once) est une famille d'algorithmes de détection d'objets en **temps réel** basée sur des réseaux de neurones convolutifs (CNN). Contrairement aux méthodes traditionnelles qui analysent une image par petites régions (comme R-CNN), YOLO traite **l'image entière en une seule passe**, ce qui le rend extrêmement rapide et efficace

Comment fonctionne YOLO ?

1.Diviser l'image en une grille

L'image est divisée en **S × S** cellules. Chaque cellule est responsable de détecter les objets dont le centre est en elle.

2. Prédire les boîtes englobantes (bounding boxes)

Pour chaque cellule, YOLO génère un certain nombre de **boîtes de détection** avec des scores de confiance.

3. Prédire la classe de l'objet

Un score de probabilité est attribué à chaque boîte pour indiquer la **classe de l'objet détecté** (ex : voiture, piéton, chien…).

4. Filtrer les résultats avec le seuil de confiance

Seules les détections avec une **forte confiance** sont conservées, et les doublons sont supprimés pour éviter plusieurs boîtes autour d'un même objet.

Application: détection d’objet en utilisant YOLOv3 et OpenCV

OpenCV (Open Source Computer Vision)

OpenCV est une bibliothèque open-source spécialisée en vision par ordinateur et en traitement d’images et de vidéos.

Elle est très utilisée dans des domaines comme la reconnaissance faciale, la détection d’objets, la vision industrielle, les systèmes ADAS, la robotique et l’intelligence artificielle.

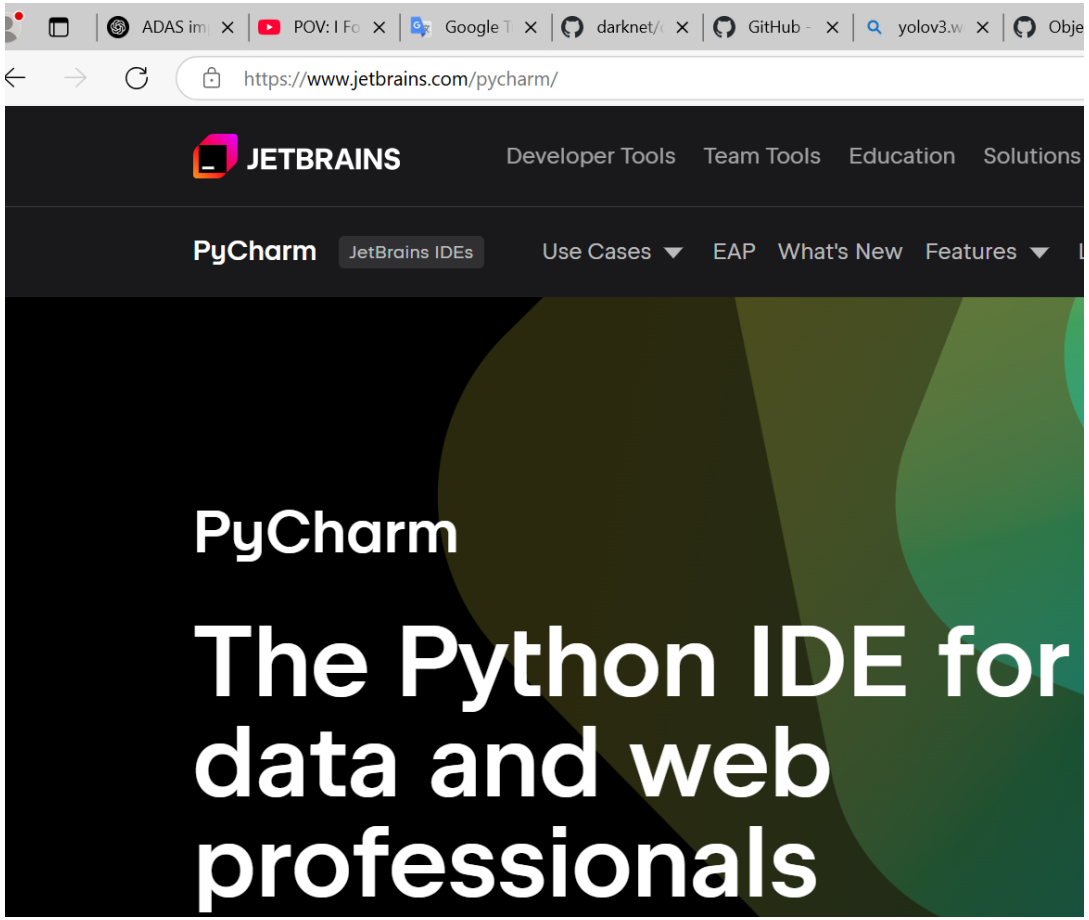
Application: étape 1

Étape 1 : installer l'environnement de travail et les packages requis

- Télécharger et installer PyCharm
- Ouvrez l'invite de commande (CMD) ou le terminal PyCharm et installez les bibliothèques Python nécessaires :

```
pip install opencv-python numpy
```

opencv-python → Utilisé pour le traitement d'images et la détection d'objets.
numpy → Utilisé pour la gestion des tableaux et des opérations mathématiques.



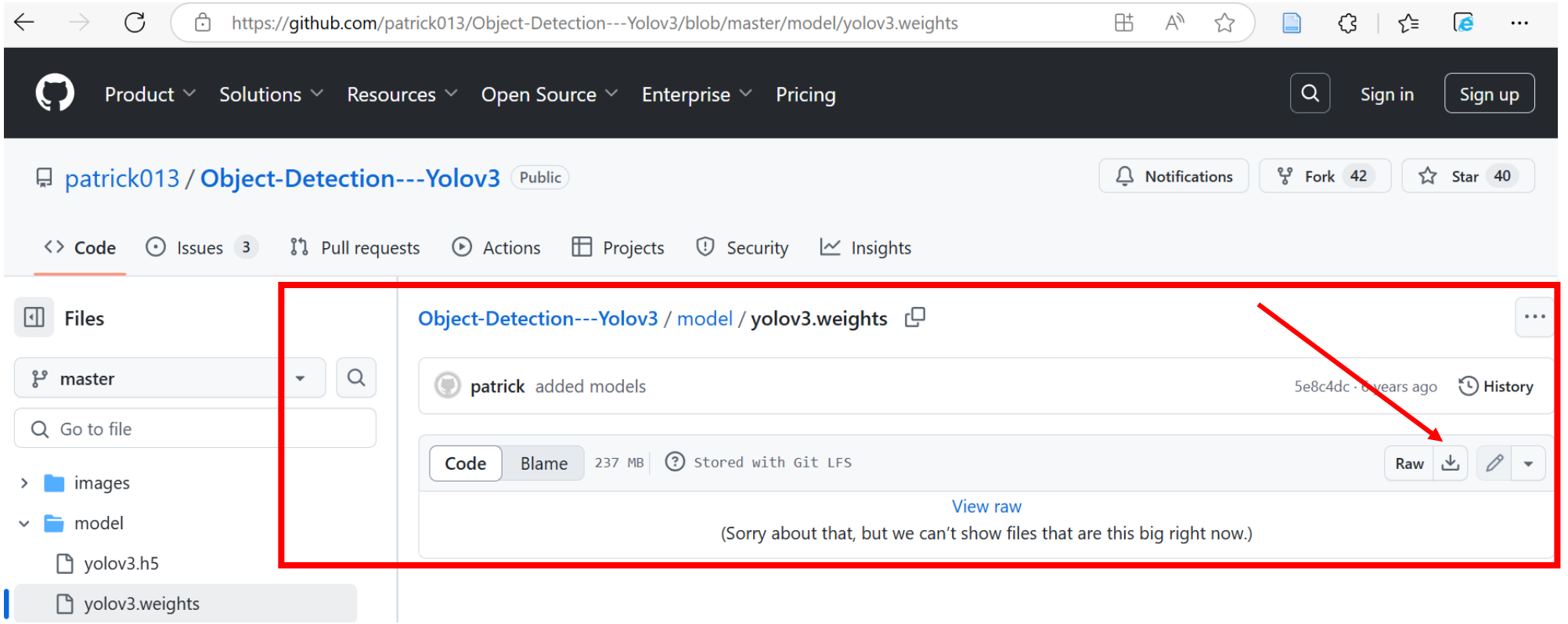
Application: étape 2

Etape 2

Téléchargez les fichiers suivants et placez-les dans un dossier appelé yolo/ dans le répertoire de votre projet :
Liens de téléchargement :

yolov3.weights: *https://github.com/patrick013/Object-Detection---Yolov3/blob/master/model/yolov3.weights*

yolov3.weights est le fichier de modèle entraîné contenant les poids de détection d'objets.



Application: étape 2 (suite)

Etape 2

Téléchargez les fichiers suivants et placez-les dans un dossier appelé yolo/ dans le répertoire de votre projet :

Liens de téléchargement : yolov3.cfg: *https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg*

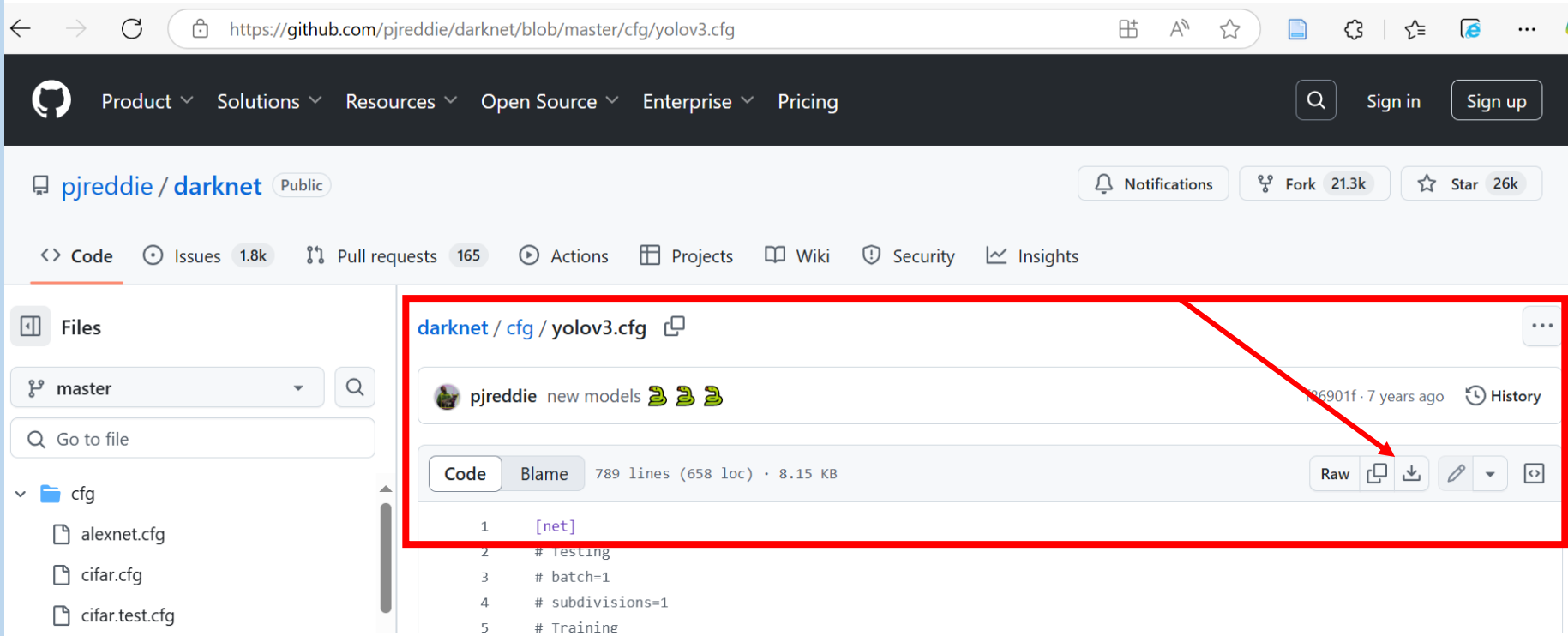
yolov3.cfg est le fichier de configuration qui définit le réseau YOLOv3.

Ce fichier contient la **configuration** du réseau. Il décrit :

L'architecture du modèle (nombre et type de couches, connexions, etc.)

Les hyperparamètres (taille des filtres, pas de convolution, normalisation, etc.)

La structure globale du réseau de neurones, qui doit être respectée lors de l'entraînement et de l'inférence.

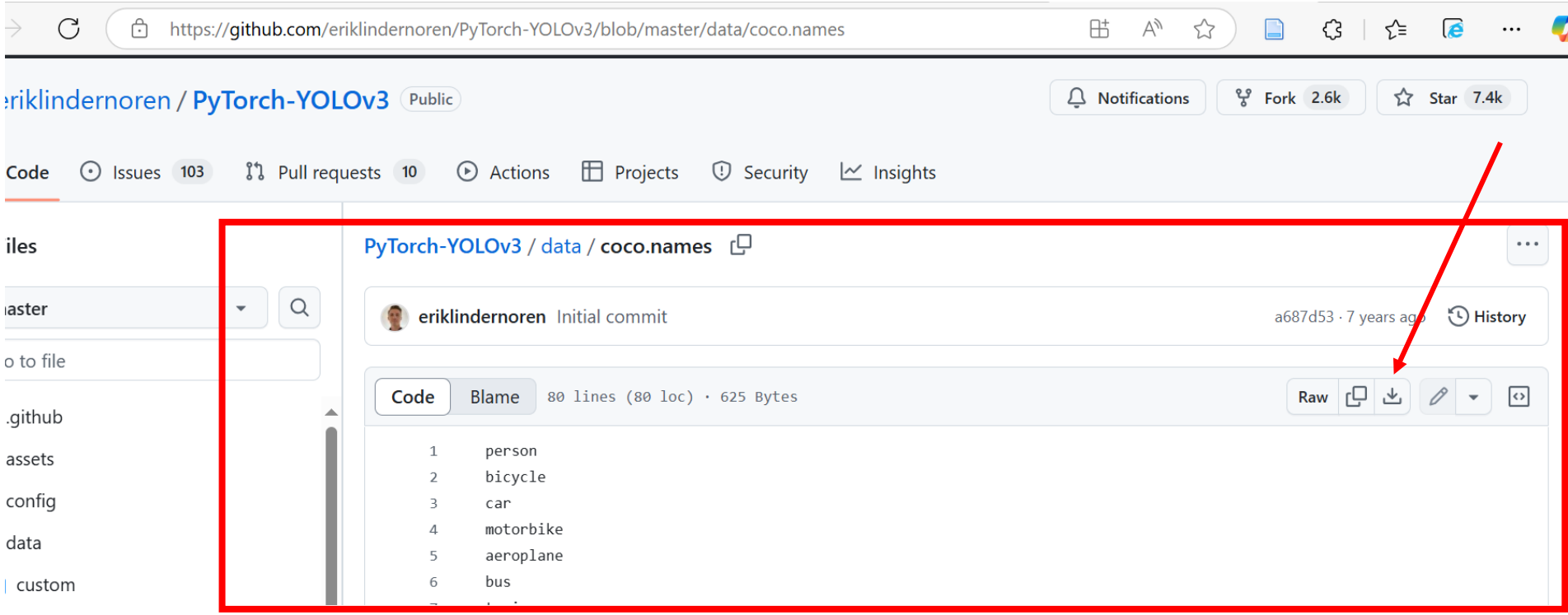


Application: étape (Suite)

Etape 2
Téléchargez les fichiers suivants et placez-les dans un dossier appelé yolo/ dans le répertoire de votre projet :
Liens de téléchargement :

coco.names: *https://github.com/eriklindernoren/PyTorch-YOLOv3/blob/master/data/coco.names*

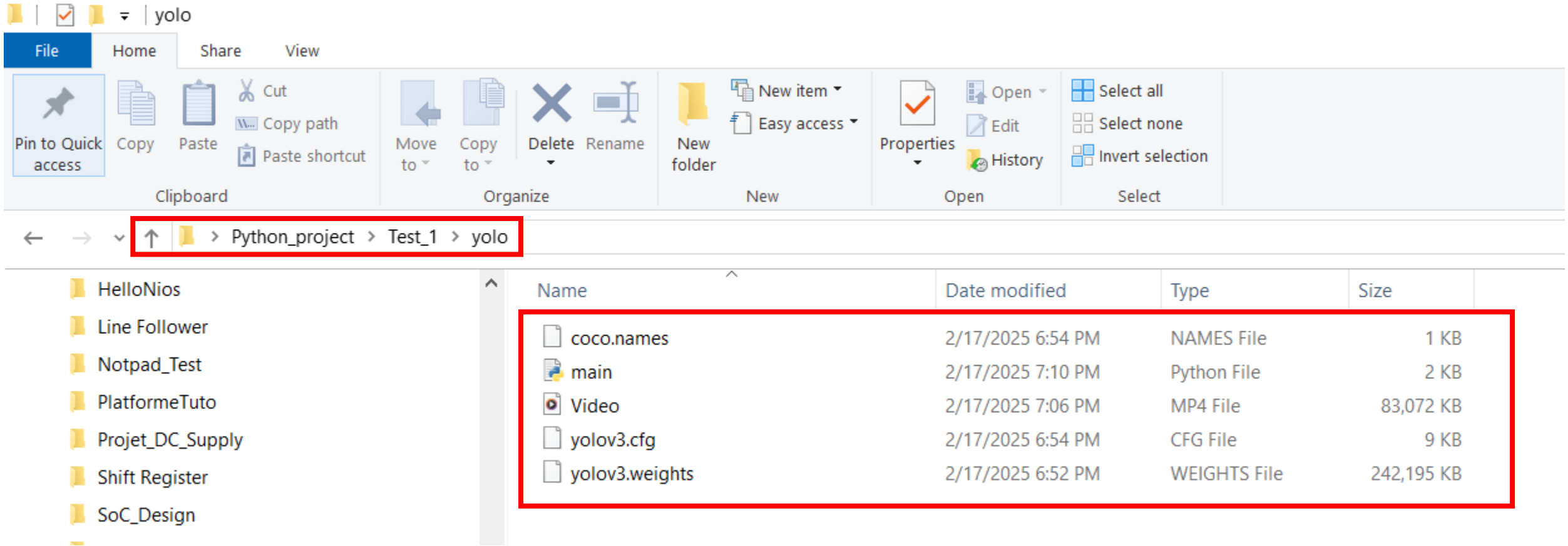
coco.names → La liste des noms d'objets que YOLO peut détecter (par exemple, personne, voiture, camion, bus, etc.).



Application: étape 3

Etape 3

- Placer tous les fichiers téléchargés dans un dossier dans votre répertoire de travail
- Créer un fichier main.py dans le même dossier

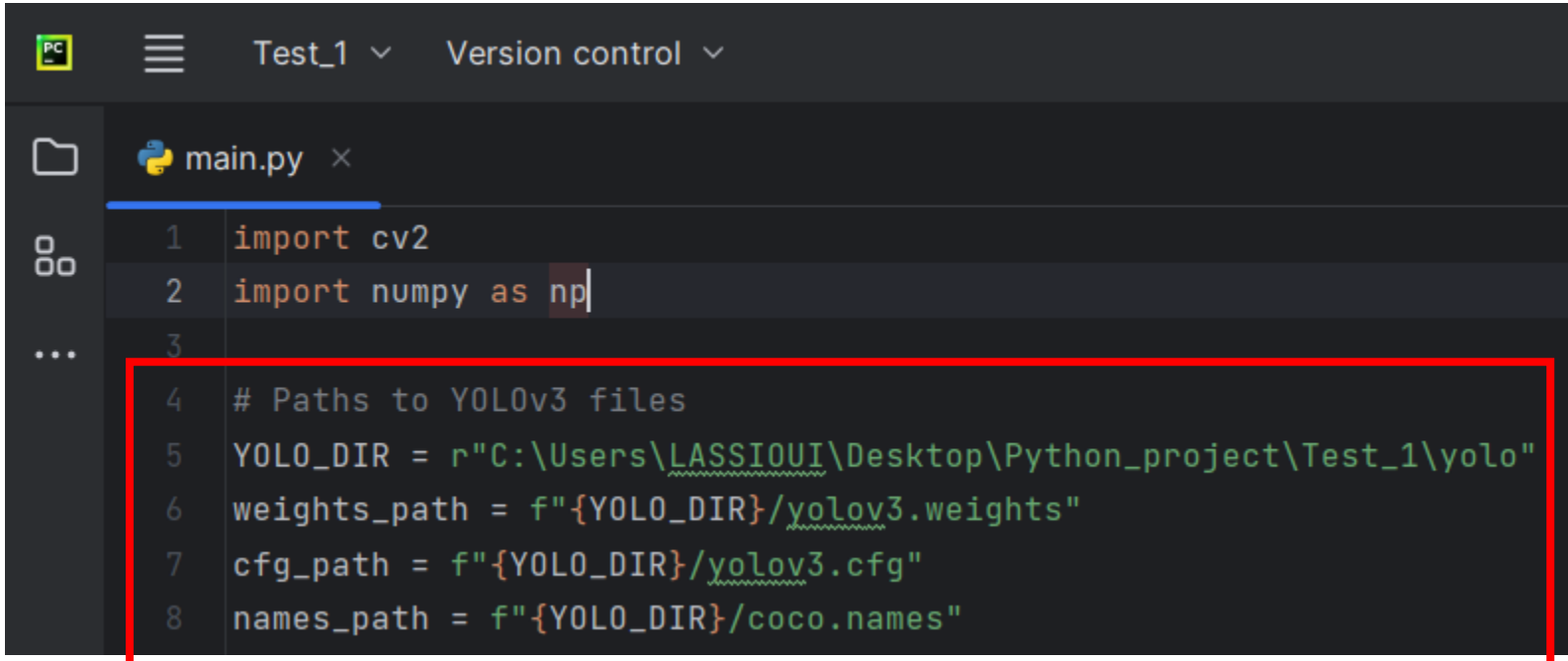


Application: étape 3

- cv2 est la bibliothèque OpenCV en Python, essentielle pour le traitement d'images et la détection d'objets.

À quoi sert cv2 ?

- Lecture de vidéos : cv2.VideoCapture()
- Traitement d'images: cv2.imread(), cv2.resize()
- Dessin de cadres de délimitation: cv2.rectangle()
- Affichage des résultats: cv2.imshow()
- Utilisation de YOLO pour la détection d'objets → cv2.dnn.readNetFromDarknet()



Spécifie les chemins des fichiers téléchargés précédemment

numpy est utilisé pour :

- Traitement des données d'image (conversion d'images en tableaux numériques)
- Opérations mathématiques (calcul des coordonnées du cadre de délimitation)
- Gestion des détections (extraction des scores de confiance et des identifiants de classe)

Application: étape 4

- Cette section du code **lit un fichier texte** contenant les **noms des classes** que YOLO peut détecter.

```
10  # Load class names
11  with open(names_path, "r") as f:
12      classes = f.read().strip().split("\n")
```

- open(names_path, "r") → Ouvre le fichier en mode lecture ("r").
- f.read()→ Lit tout le contenu du fichier dans une seule chaîne de caractères.
- .strip() → Supprime les espaces et les sauts de ligne inutiles au début et à la fin.
- .split("\n") → Divide la chaîne en une liste où chaque élément est un nom de classe (chaque ligne devient un élément de la liste).

```
person
bicycle
car
motorbike
bus
truck
```

- Après l'exécution de la ligne de code

```
["person", "bicycle", "car", "motorbike", "bus", "truck"]
```

Cette liste classes est ensuite utilisée pour afficher le nom des objets détectés :

```
# Exemple : Si YOLO détecte une voiture (classe index 2)
class_id = 2
print("Objet détecté:", classes[class_id])  # Output: "Objet détecté: car"
```

Application: étape 5

- Ces lignes de code utilisent OpenCV et le module **Deep Neural Network (DNN)** pour charger un modèle **YOLO** (You Only Look Once) destiné à la détection d'objets.

```
14  # Load YOLO model
15  net = cv2.dnn.readNetFromDarknet(cfg_path, weights_path)
16  layer_names = net.getLayerNames()
17  output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
18
```

Net = cv2.dnn.readNetFromDarknet(cfg_path, weights_path) charge un réseau neuronal YOLO à partir de :

- cfg_path : le chemin vers le fichier **de configuration** (.cfg), qui contient l'architecture du réseau.
- weights_path : le chemin vers le fichier **des poids** (.weights), qui stocke les valeurs des paramètres appris.

Le réseau est maintenant stocké dans la variable net et peut être utilisé pour **l'inférence**.

layer_names = net.getLayerNames()

- net.getLayerNames() récupère la liste de toutes les couches du réseau sous forme de chaînes de caractères.
- YOLO a plusieurs couches, dont certaines produisent des **sorties détectant des objets**.

Application: étape 5 (suite)

- Ces lignes de code utilisent OpenCV et le module **Deep Neural Network (DNN)** pour charger un modèle **YOLO** (You Only Look Once) destiné à la détection d'objets.

```
14  # Load YOLO model
15  net = cv2.dnn.readNetFromDarknet(cfg_path, weights_path)
16  layer_names = net.getLayerNames()
17  output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
18
```

output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

- net.getUnconnectedOutLayers() retourne une liste **d'indices** des couches qui produisent des **sorties finales** du réseau.
- YOLO a généralement **trois couches de sortie** (correspondant aux trois échelles de détection pour objets de différentes tailles).
- Comme les indices OpenCV commencent à **1** et que Python utilise un **index basé sur 0**, il faut faire i - 1 pour accéder correctement aux noms des couches.
- output_layers contiendra les **noms** des couches responsables des détections (par exemple ['yolo_82', 'yolo_94', 'yolo_106'] pour YOLOv3).

Application

❑ Ces lignes de code serts à lire une vidéo image par image en utilisant OpenCV.

```
21  cap = cv2.VideoCapture("Video_2.mp4")  # Change this to your video file
22
23  while cap.isOpened():
24      ret, frame = cap.read()
25      if not ret:
26          break
```

Cap = cv2.VideoCapture est utilisé pour ouvrir le fichier vidéo nommé "Video_2.mp4".

· La variable cap représente l'objet qui gère la capture de la vidéo.

· La boucle while cap.isOpened(): s'assure que la vidéo est correctement ouverte.

· À chaque itération, cap.read() lit une nouvelle image (ou frame) de la vidéo.

- ret est un booléen qui indique si la lecture a réussi.
- frame contient l'image lue.

· Si ret est False (c'est-à-dire que la lecture a échoué ou que la vidéo est terminée), la boucle est interrompue avec break.

Application

Ce code prépare chaque frame d'une vidéo pour la détection d'objets par YOLO en la normalisant, redimensionnant et convertissant en format adapté, puis en passant cette image dans le réseau pour obtenir les détections.

```
28      height, width = frame.shape[:2]
29
30      # Convert frame to blob for YOLO
31      blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, crop=False)
32      net.setInput(blob)
33      detections = net.forward(output_layers)
```

· La fonction cv2.dnn.blobFromImage convertit l'image en un **blob** : une structure de données (tableau multi-dimensionnel) qui est adaptée à l'entrée du réseau de neurones.

· **Paramètres importants :**

- 1/255.0 : Normalise les valeurs de pixels (allant initialement de 0 à 255) pour qu'elles soient entre 0 et 1.
- (416, 416) : Redimensionne l'image à une taille de 416x416 pixels, taille attendue par le modèle YOLO.
- swapRB=True : Permet d'inverser les canaux rouge et bleu car OpenCV lit l'image en format BGR alors que la plupart des réseaux (dont YOLO) attendent un format RGB.
- crop=False : Indique de ne pas recadrer l'image après redimensionnement.

Application

Ce code prépare chaque frame d'une vidéo pour la détection d'objets par YOLO en la normalisant, redimensionnant et convertissant en format adapté, puis en passant cette image dans le réseau pour obtenir les détections.

```
28      height, width = frame.shape[:2]
29
30      # Convert frame to blob for YOLO
31      blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, crop=False)
32      net.setInput(blob)
33      detections = net.forward(output_layers)
```

Passage du blob dans le réseau

net.setInput(blob)

Le blob ainsi créé est défini comme l'entrée du réseau grâce à setInput.

· **Exécution de l'inférence pour obtenir les détections**

detections = net.forward(output_layers)

output_layers correspond aux couches de sortie que l'on a préalablement définies.

Le résultat (detections) contient les informations sur les objets détectés (comme les classes, les scores de confiance).

Application

Ce segment de code traite les résultats du modèle YOLO pour extraire les informations sur les objets détectés.

```
35      for output in detections:
36          for detection in output:
37              scores = detection[5:]
38              class_id = np.argmax(scores)
39              confidence = scores[class_id]
40
```

for output in detections:
La boucle parcourt chacune des sorties générées par YOLO.

for detection in output:
Chaque detection est un vecteur contenant plusieurs informations. En général, les premières positions correspondent aux classes et à la confiance et les scores pour chaque classe.

scores = detection[5:]
Cette ligne sélectionne les éléments du vecteur detection à partir de l'indice 5, qui représentent les **scores d'appartenance** pour chaque classe prédite par le modèle.

class_id = np.argmax(scores)
np.argmax(scores) retourne l'indice du score le plus élevé dans le vecteur scores.

confidence = scores[class_id]
Une fois la classe identifiée, cette ligne extrait la **valeur de confiance** associée à cette classe, indiquant à quel point la détection est fiable.

✓ Ce segment de code parcourt toutes les détections produites par le modèle YOLO, extrait les scores pour chaque classe, identifie la classe avec le score le plus élevé et récupère la confiance correspondante.

Application

Ce morceau de code applique un **seuil de confiance**, extrait les **coordonnées des objets détectés**, puis **dessine des boîtes englobantes (bounding boxes) avec des étiquettes** sur l'image.

```
40
41          if confidence > 0.5:  # Confidence threshold
42              center_x, center_y, w, h = (detection[:4] * [width, height, width, height]).astype("int")
43
44              x = int(center_x - w / 2)
45              y = int(center_y - h / 2)
46
47              label = f"{classes[class_id]}: {confidence:.2f}"
48              cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
49              cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
50
51      cv2.imshow("YOLO Detection", frame)
```

if confidence > 0.5:
Cette condition s'assure que seule une détection ayant une confiance supérieure à 50 % est prise en compte.
Cela permet de ne garder que les objets détectés de manière fiable.

center_x, center_y, w, h = (detection[:4] * [width, height, width, height]).astype("int")
detection[:4] contient :

- center_x, center_y : les **coordonnées du centre** de la boîte englobante **normalisées** (valeurs entre 0 et 1).
- w, h : la **largeur et la hauteur** de la boîte englobante, également normalisées.

* [width, height, width, height] ramène ces valeurs à l'échelle de l'image.

astype("int") convertit les valeurs en **entiers**

Application

Ce morceau de code applique un **seuil de confiance**, extrait les **coordonnées des objets détectés**, puis **dessine des boîtes englobantes (bounding boxes) avec des étiquettes** sur l'image.

```
40
41          if confidence > 0.5:  # Confidence threshold
42              center_x, center_y, w, h = (detection[:4] * [width, height, width, height]).astype("int")
43
44              x = int(center_x - w / 2)
45              y = int(center_y - h / 2)
46
47              label = f"{classes[class_id]}: {confidence:.2f}"
48              cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
49              cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
50
51      cv2.imshow("YOLO Detection", frame)
```

x = int(center_x - w / 2)
y = int(center_y - h / 2)
OpenCV dessine les rectangles avec (x, y) correspondant au **coin supérieur gauche**, alors que YOLO donne le **centre** de la boîte.
Ces lignes calculent donc les coordonnées du **coin supérieur gauche** en soustrayant la moitié de la largeur (w/2) et de la hauteur (h/2) à center_x et center_y

label = f"{classes[class_id]}: {confidence:.2f}"
classes[class_id] donne le **nom de la classe** (par exemple, "voiture", "personne", "chien").
confidence:.2f formate la confiance en **deux décimales** (exemple 85%)

cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
cv2.rectangle dessine une boîte verte ((0, 255, 0)) autour de l'objet détecté.
2 est l'épaisseur du contour.

Application

Ce morceau de code applique un **seuil de confiance**, extrait les **coordonnées des objets détectés**, puis **dessine des boîtes englobantes (bounding boxes) avec des étiquettes** sur l'image.

```
40
41          if confidence > 0.5:  # Confidence threshold
42              center_x, center_y, w, h = (detection[:4] * [width, height, width, height]).astype("int")
43
44              x = int(center_x - w / 2)
45              y = int(center_y - h / 2)
46
47              label = f"{classes[class_id]}: {confidence:.2f}"
48              cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
49              cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
50
51      cv2.imshow("YOLO Detection", frame)
```

cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

- cv2.putText écrit le **nom de l'objet détecté + la confiance** juste au-dessus de la boîte.
- (x, y - 10): place le texte légèrement au-dessus du coin supérieur gauche.
- cv2.FONT_HERSHEY_SIMPLEX, 0.5: police et taille du texte.
- (0, 255, 0), 2: couleur verte et épaisseur de texte.

cv2.imshow("YOLO Detection", frame)

- Cette ligne affiche l'image avec les **boîtes englobantes et les labels** dans une fenêtre nommée "YOLO Detection".
- L'image est mise à jour à chaque frame de la vidéo.

Application

Ce morceau de code permet si l'utilisateur appuie sur la touche «q » d'interrompre la boucle de traitement de la vidéo.
De terminer l'accès à la vidéo et la libérer pour pouvoir l'utiliser dans d'autres tâches et finalement fermer toutes les fenêtres crées par OpenCV.

```
50
51      cv2.imshow("YOLO Detection", frame)
52      if cv2.waitKey(1) == ord("q"):
53          break
54
55  cap.release()
56  cv2.destroyAllWindows()
57
```

Application: programme total

main.py

```python
import cv2
import numpy as np

# Paths to YOLOv3 files
YOLO_DIR = r"C:\Users\LASSIOUI\Desktop\Python_project\Test_1\yolo"
weights_path = f"{YOLO_DIR}/yolov3.weights"
cfg_path = f"{YOLO_DIR}/yolov3.cfg"
names_path = f"{YOLO_DIR}/coco.names"

# Load class names
with open(names_path, "r") as f:
    classes = f.read().strip().split("\n")

# Load YOLO model
net = cv2.dnn.readNetFromDarknet(cfg_path, weights_path)
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

# Open video file

cap = cv2.VideoCapture("Video_2.mp4")  # Change this to your video file

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    height, width = frame.shape[:2]

    # Convert frame to blob for YOLO
    blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, crop=False)
    net.setInput(blob)
    detections = net.forward(output_layers)

    for output in detections:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]

            if confidence > 0.5:  # Confidence threshold
                center_x, center_y, w, h = (detection[:4] * [width, height, width, height]).astype("int")

                x = int(center_x - w / 2)
                y = int(center_y - h / 2)

                label = f"{classes[class_id]}: {confidence:.2f}"
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(frame, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    cv2.imshow("YOLO Detection", frame)
    if cv2.waitKey(1) == ord("q"):
        break

cap.release()
cv2.destroyAllWindows()
```