

# Voir autrement, bouger librement, vivre pleinement !

«Comment développer un dispositif abordable  
qui permettrait de normaliser le mouvement  
au sein d'une ville pour les personnes  
aveugles et malvoyantes ? »

académie  
Nancy-Metz



Lycée Georges  
de la Tour

BOUTOUILI Noor  
GLAD Martin  
POCHON Lily  
RAPINE Edessa



Sciences à l'École



CONCOURS  
CGÉNIAL  
Collège  
Lycée

FONDATION  
CGÉNIAL

# NOTRE EQUIPE



## Au design et à la rédaction :

**Edessa RAPINE**



**Lily POCHON**



## À la programmation :

**Noor BOUTOUILI**



**Martin GLAD**



# Introduction

## Nos objectifs :

D'après l'organisation mondiale de la santé, la déficience visuelle a de graves répercussions sur la qualité de vie des populations adultes. Les adultes ayant une déficience visuelle peuvent avoir des taux plus élevés de dépression, d'anxiété, et de suicide. Ce handicap peut contribuer à l'isolement social, et à l'absence de soutien.

D'un point de vue sécuritaire en France, il y a 100 000 à 150 000 feux de signalisation, mais seulement 35% (soit 45 000) sont équipés de dispositifs sonores pour aider les personnes malvoyantes ou aveugles. Ces dispositifs permettent une traversée plus sûre en émettant un signal sonore lors du passage piéton vert, mais ils peuvent être inefficaces si le bruit ambiant est trop fort. Les feux sonores sont également plus coûteux que les feux classiques : leur prix varie entre 25 000 et 40 000 euros, soit environ 113% de plus qu'un feu standard. Un feu avec volume ajustable et système tactile ou vibrant, plus adapté, coûte entre 35 000 et 50 000 euros, soit environ 180% plus cher.

D'un point de vue financier, notre dispositif revient à moins de 500 euros, ce qui le rend très intéressant et plus abordable, même si celui-ci ne permet d'équiper qu'une seule personne.

Notre objectif est donc l'intégration à un prix abordable de personnes atteintes d'un handicap ou d'une incapacité visuelle. Ces lunettes pourraient améliorer considérablement la qualité de vie des personnes malvoyantes en leur offrant une plus grande indépendance et une meilleure intégration dans leur environnement quotidien, en plus d'une sécurité accrue.

Notre projet consiste à programmer un Raspberry Pi 5 et à utiliser le flux vidéo d'une caméra pour interpréter l'environnement urbain. Nous nous sommes d'abord focalisés sur l'identification des feux de signalisation et des passages piétons, pour une sécurité accrue des personnes aveugles ou malvoyantes. Un signal sonore interprétant l'identification de ces infrastructures leur sera transmis et leur permettra de mieux interagir en société en encourageant le déplacement individuel dans des villes peu, voire pas adaptés, pour leur handicap.

## Définitions :



OpenCV (Open Source Computer Vision Library) est une bibliothèque open-source spécialisée en vision par ordinateur et en apprentissage automatique. Les projets open-source permettent à n'importe qui de les utiliser, de les modifier et de les redistribuer librement.

OpenCV peut être utilisée pour le traitement d'images en temps réel, la détection d'objets, la reconnaissance faciale, et bien d'autres applications. Dans notre cas, nous l'avons utilisé pour repérer des objets bien particuliers : les feux de signalisation, les passages piétons et les obstacles.

Le Raspberry Pi est un nano-ordinateur monocarte, c'est-à-dire un ordinateur réduit à l'essentiel, tenant sur une seule carte électronique de la taille d'une carte de crédit. Il fonctionne avec un processeur ARM, comme ceux des smartphones, et possède des ports USB, HDMI et GPIO qui permettent de le connecter à divers périphériques.



Son principal intérêt est de rendre l'informatique accessible à tous, à un coût réduit, tout en encourageant l'apprentissage de la programmation et des systèmes informatiques. En effet son poids léger fait de lui un nano-ordinateur facilement transportable, ce qui nous intéresse grandement dans notre projet, pour pouvoir créer un modèle de lunette compact et léger, pour une utilisation la plus pratique possible.

## Questionnement :

A ce stade de notre réflexion, nous nous sommes posés de nombreuses questions :

- Est-ce que la détection sera fiable ?
- Que voulons-nous repérer grâce à ces lunettes ? Que pouvons-nous repérer avec un programme facile à mettre en place ?
- La caméra aura-t-elle une portée assez large ? Dans le cas contraire, dans quel modèle devrions-nous investir pour corriger cette erreur ?
- Est-ce que la caméra sera assez petite et légère pour tenir sur des lunettes et ne pas être désagréable ?
- Quel type de signal allons-nous transmettre à la personne (à travers l'oreillette) ?
- Comment réguler le flux d'information qui arrivera à l'oreillette ?
- La caméra arrivera-t-elle à capter les objets assez rapidement ?
- Faut-il rajouter un voyant aux lunettes pour prévenir qu'elles filment (d'un point de vue éthique) ?
- Sous quelles conditions les lunettes ne fonctionnent pas correctement (luminosité, contraste des couleurs...) ?



## Premières réflexions

Le projet consiste à interpréter l'environnement urbain en utilisant la caméra. Le flux vidéo est ensuite envoyé au dispositif informatique autonome. Il faudra donc programmer la Raspberry Pi 5 avec son accélérateur d'IA HAT pour interpréter les informations reçues en direct.

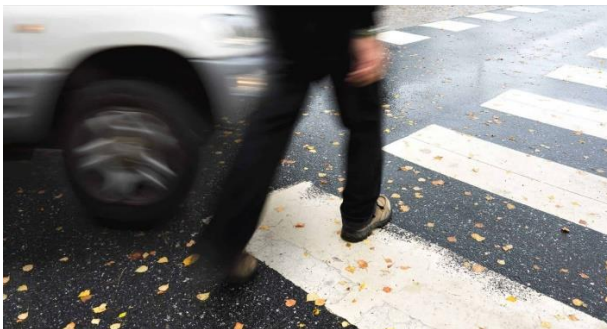
Bandeau

Caméra

Dispositif informatique autonome :  
raspberry Pi 5 avec son accélérateur d'IA



Lors de notre projet, nous nous sommes d'abord focalisés sur l'identification des feux de signalisation et des passages piétons, car ce sont deux éléments non indiqués sur des applications comme 'maps', mais qui sont pourtant essentiels à la sécurité des personnes malvoyantes ou aveugles lors de leurs déplacements.



Bien que notre dispositif ne puisse empêcher tous les accidents, il permettra de protéger au mieux la traversée des routes en milieu urbain.

Une fois que notre dispositif a identifié le passage protégé et/ou le feu de signalisation qui est bien vert pour les piétons, comment prévenir la personne ?

Nous avons pensé dans un premier temps utiliser un signal sonore. Mais il y a de forte chance que la personne utilise déjà des oreillettes pour recevoir les informations provenant de l'application 'Maps' par exemple. Notre système ne doit donc pas interférer avec des écouteurs. De plus, en ville, la pollution sonore et les bruits environnants peuvent atteindre jusqu'à 110 dB ! Il nous paraissait donc impossible de générer un système sonore pouvant palier à tous ces problèmes.

Nous avons alors décidé d'avertir l'utilisateur grâce à une vibration du boîtier, la Raspberry pi 5 étant compatible avec des systèmes de buzzer. Il nous reste ensuite la question du message en lui-même, comment avertir l'utilisateur ? Nous avons plusieurs idées en tête qu'il nous faudra tester avant de choisir la plus efficace. Pour l'instant la solution la plus simple nous semble être l'émission de deux vibrations distinctes : une plus rapide et puissante pour avertir d'un danger imminent et une autre plus faible et longue pour signaler que le passager peut traverser.

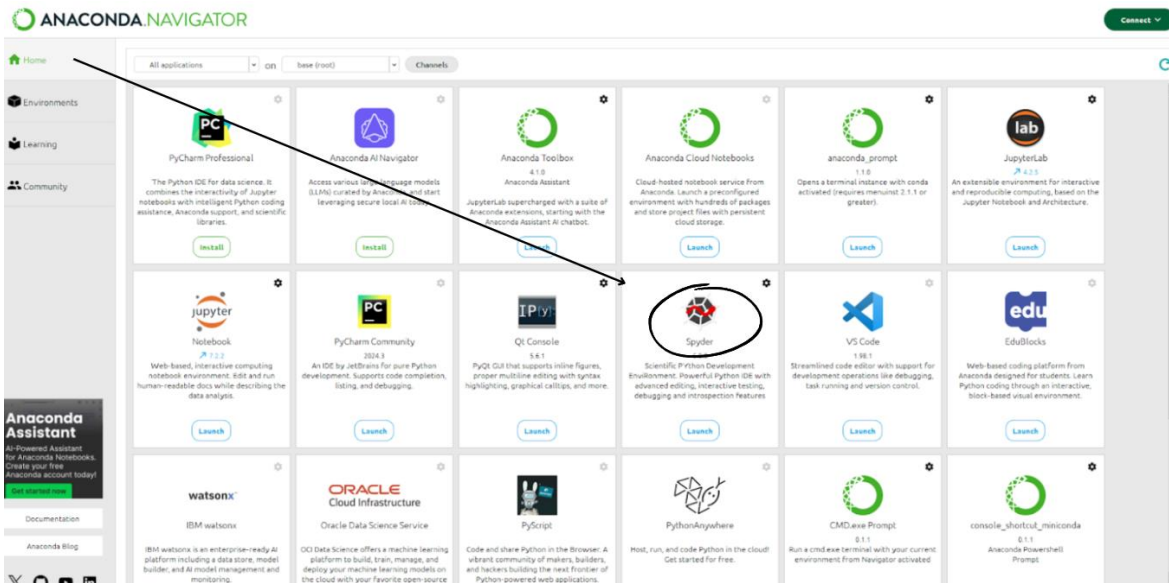
## **Installation des outils de développement**

Maintenant que le contexte est posé, nous allons commencer à exploiter le flux vidéo de la caméra. Nous nous sommes documentés et nous avons été orientés pour utiliser une bibliothèque d'outils spécialisée dans la vision : OpenCV.

Après une longue réflexion et quelques essais, nous avons opté pour une installation plus globale, c'est-à-dire une installation de cette bibliothèque dans un environnement de travail complet qui intègre tous les outils de développement dont nous aurons besoin : OpenCV + Python + IDE pour python. Nous avons donc procédé à l'installation d'Anaconda.

Pour le moment, nous avons un ensemble opérationnel avec un environnement sous Windows, mais à terme, nous pensons travailler avec Linux. Comme nous avons choisi de l'informatique embarqué avec une bibliothèque et des logiciels libres de droit, autant utiliser également un environnement aussi libre de droit comme Linux.

Sous Anaconda, nous obtenons l'interface graphique suivante :



Depuis cet intégrateur d'environnement nous avons accès à tous nos outils, et il suffit d'utiliser l'IDE (Integrated Development Environment) Spyder.

Maintenant que nous avons choisi notre système informatique et les différents outils que nous allons utiliser pour effectuer le développement, il ne reste plus que l'essentiel, à savoir, le développement des différents programmes permettant d'aider ces personnes.



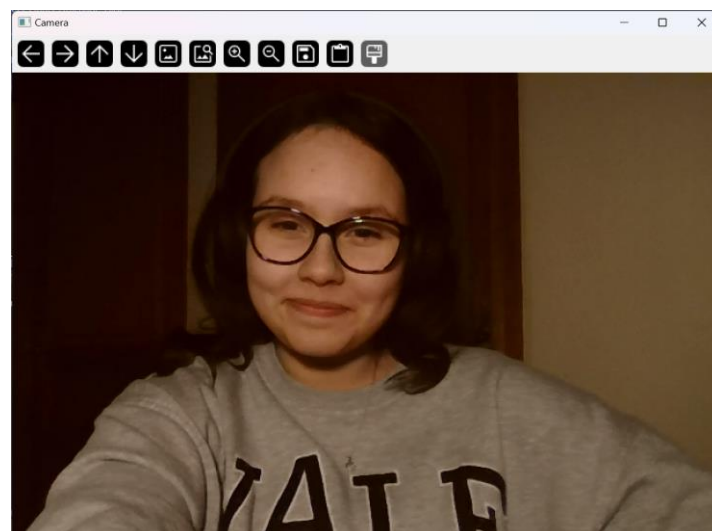
## Détection des feux et des passages piétons

Au commencement de notre projet, nous n'avions pas les connaissances nécessaires au développement des programmes souhaité sur OpenCV. Nous avons dû apprendre à utiliser cette bibliothèque OpenCV, qui est très complexe et qui demande de la technicité quant à sa mise en œuvre. Dans cette partie nous aborderons essentiellement la détection des couleurs.

**1 - Affichage du flux vidéo capturé par la caméra dans une fenêtre :**

```
Fichier  Édition  Recherche  Source  Exécution  Débugger  Console  Projets  Outils  Affichage  Aide
code_OpenCv.py  color_detection.py  color_detect_3.py
1  #-*- coding: utf-8 -*-
2  """
3  Created on Wed Nov 27 15:19:15 2024
4
5  @author: Eleve
6  """
7
8  import cv2 #utiliser la bibliothèque OpenCv
9
10
11 # choix de la caméra
12 vid = cv2.VideoCapture(0)
13
14 #RET est un booléen, et F pour frame qui représente le cadre de la vidéo
15 while True :
16     RET, F = vid.read() #permet de lire un nouveau cadre dans la vidéo
17     cv2.imshow("Direct Noor",F) #affiche le cadre vidéo dans une fenêtre
18     KEY = cv2.waitKey(1) #Attend la touche pressée
19     if KEY == ord("q"):
20         break # arrête la boucle while qui montre le flux vidéo en direct
21     vid.release() #libère la mémoire associée à la webcam (fermer le flux vidéo)
22     cv2.destroyAllWindows() # Détruit toutes les fenêtres
23
24
```

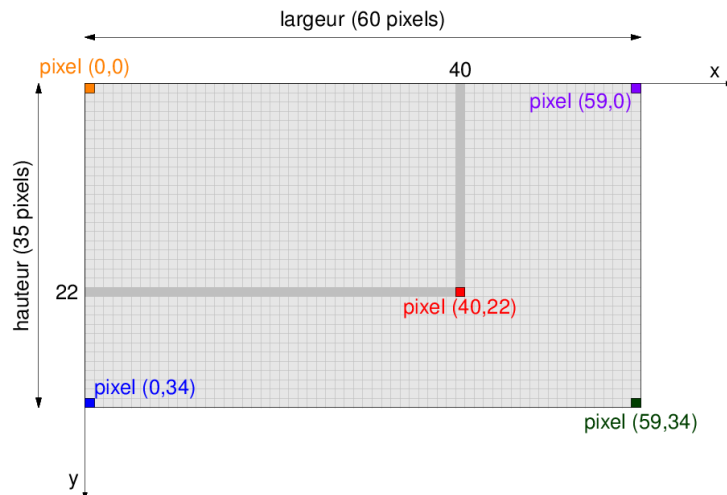
Ce programme permet seulement d'afficher le flux vidéo de la caméra et de l'afficher dans une fenêtre :



Après avoir mis en œuvre la technique permettant d'acquérir des images, voici quelques exemples qui nous ont fait comprendre comment détecter une couleur.

## 2 - Qu'est-ce qu'une image ?

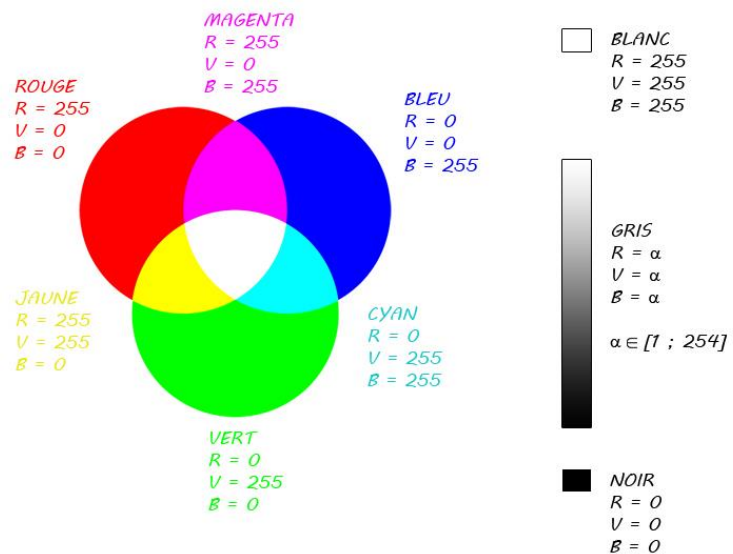
Une image est un ensemble de points élémentaire appelé pixel. La dalle d'un écran peut être représentée sous forme matricielle. Dans l'exemple ci-dessous, nous avons un écran de 60x35 :



([https://nsimichelet91.github.io/snt/Theme1\\_Image\\_numerique/cours/](https://nsimichelet91.github.io/snt/Theme1_Image_numerique/cours/))

Pour être affiché sur un écran, chaque pixel doit posséder un certain nombre de caractéristiques. Les écrans ont pour la plupart besoin des paramètres : R (Rouge), V (Vert) et B (Bleu). Donc lorsque nous lisons une image avec un traitement informatique, nous obtenons une image au format RVB (BGR en anglais).

De plus, chaque couleur est codée avec une valeur de 0 à 255. Cela donne une possibilité de  $256 \times 256 \times 256 = 16\,777\,216$  couleurs différentes pour chaque pixel. Voici ci-dessous des exemples de valeurs pour certaines couleurs



([https://nsimichelet91.github.io/snt/Theme1\\_Image\\_numerique/cours/](https://nsimichelet91.github.io/snt/Theme1_Image_numerique/cours/))

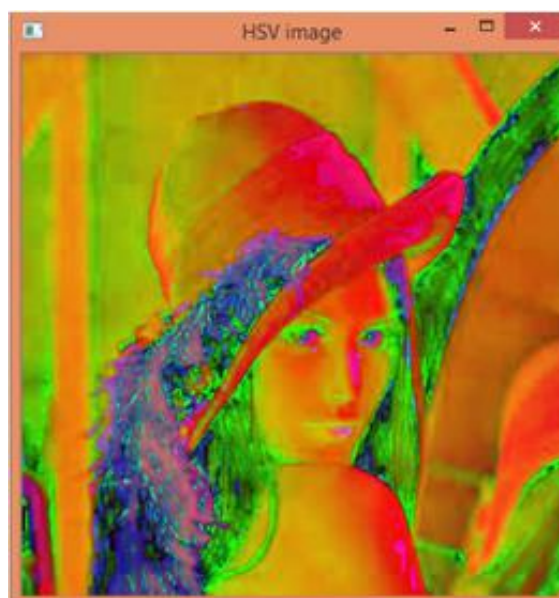
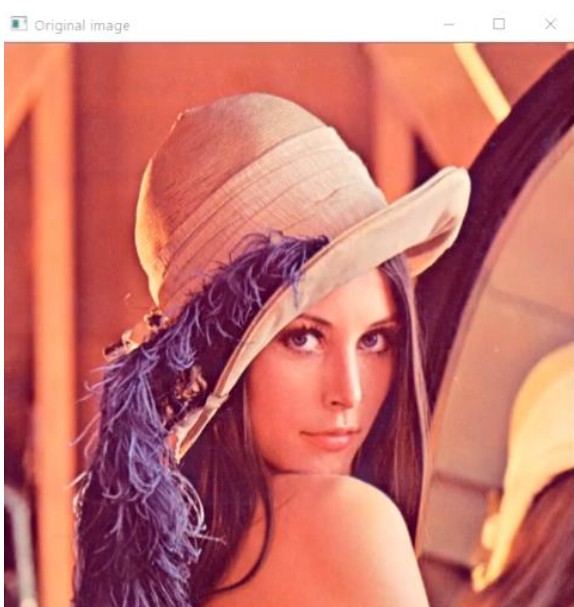
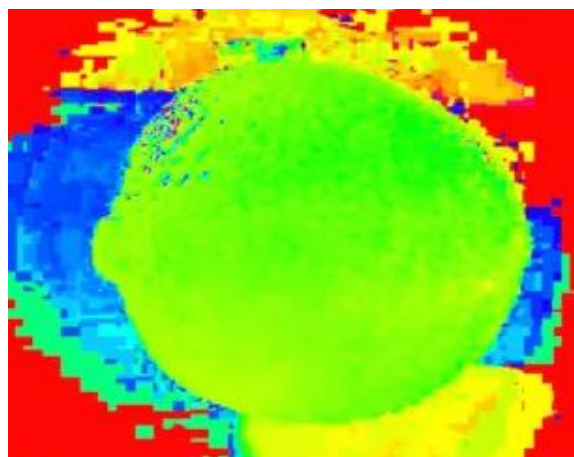
Le modèle RGB a été techniquement élaboré pour les différents écrans. Cependant, il existe d'autres modèles de représentation des couleurs dans lequel la luminosité ambiante interfère nettement moins la détection de celle-ci. C'est notamment le cas pour le modèle HSV.

Voici des exemples montrant les différences entre ces deux modèles :

Image codée en RGB :



Image codée en HSV :



Comme vous pouvez le voir, le modèle HSV a pu mettre clairement en évidence les limites du citron, en plus, elle a été capable d'éliminer les effets d'éclairage sur le citron lui-même. Il en est de même avec l'image du portrait ou des zones de couleurs apparaît. Mais pour cette image ou la palette des couleurs est moins vaste, le résultat obtenu doit être un peu plus interprété.

Une autre observation est qu'il y a une rougeur autour du citron, pour en comprendre la raison, il faut savoir que pour le modèle HSV, il n'y a que des couleurs, pas de teintes blanche/gris/noir, ce qui signifie que si l'image a une partie qui est incolore (blanc, gris, noir), alors sa représentation en HSV sera presque aléatoire, parce que la saturation et la valeur vont détruire la couleur résultant de la couleur grisâtre que nous voyons.

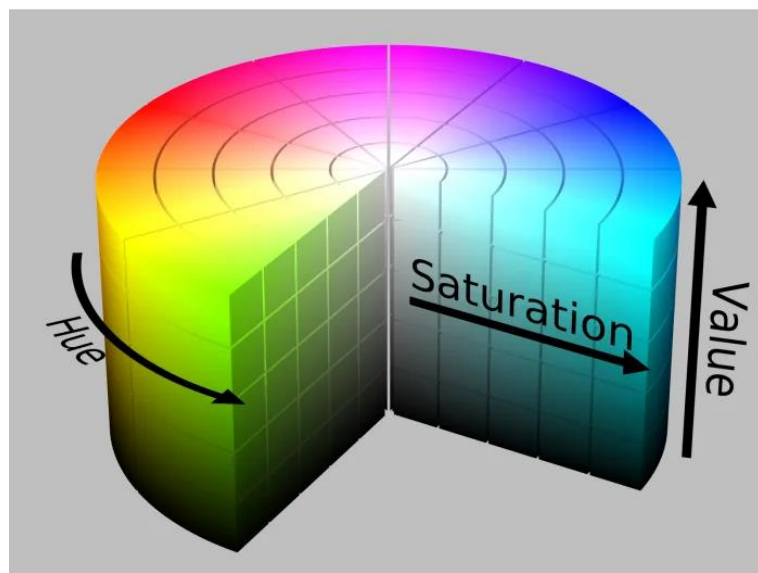
## Que signifie l'acronyme HSV ?

H (Hue) : représente les différentes couleurs disponibles et nous pouvons accéder à différentes couleurs en donnant des valeurs différentes pour la teinte sous forme d'angle.

S (Saturation) : représente la quantité de couleur/pigment. Cela indique par exemple, que pour une même couleur, l'objet est représenté avec différente nuance en fonction de la lumière ambiante.

V (Valeur) : représente la valeur de la brillance de la couleur. Cela indique par exemple, que pour une même couleur, l'objet est représenté avec différente intensité en fonction de la lumière ambiante.

Espace de couleur HSV



Wikipédia

C'est donc le modèle HSV qui a été choisi pour la recherche d'une couleur sur une image. Le jeu d'instruction nécessaire est la suite : `cv2.cvtColor` (converti le modèle d'une couleur en un autre modèle de couleur). Une mise œuvre simple serait la suivante :

```
importations cv2

Chargement d'une image
image de cv2.imread('path-to-image.jpg')

Conversion de BGR en RGB
rgb-image : cv2.cvtColor (image, cv2. COLOR_BGR2RGB)
```

Nous allons maintenant voir comment cela a été utilisé dans nos différents essais.

### 3 - Détection d'une seule couleur : ici le jaune

Nous avons commencé par essayer d'adapter des applications nous permettant, par la suite, de détecter au mieux les couleurs qui nous intéressent. A savoir le rouge, l'orange et le vert.

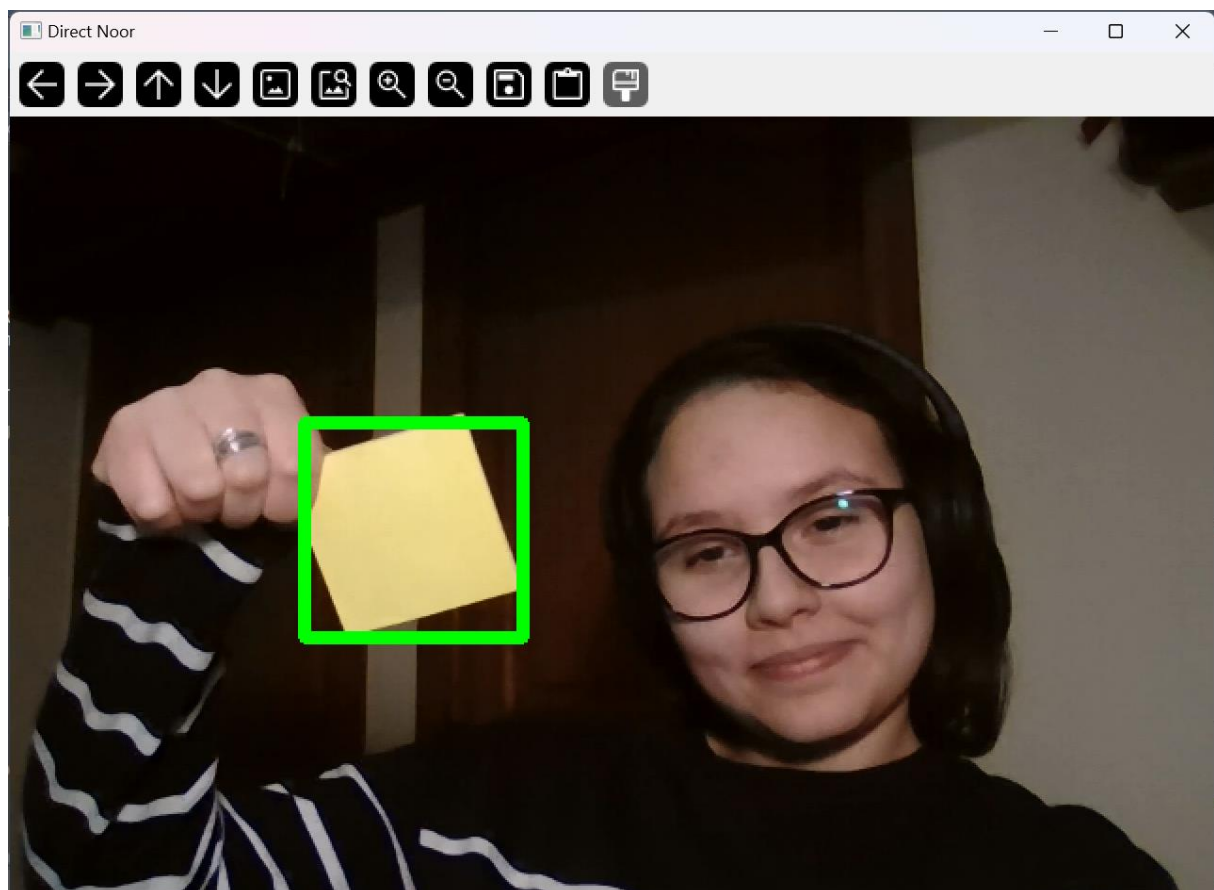
Le programme ci-dessous permet d'identifier dans le flux vidéo la couleur jaune.

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat Dec 14 18:27:01 2024
4
5  @author: Eleve
6  """
7  import cv2
8  import numpy as np
9  from PIL import Image
10
11
12  yellow = [0,255,255]
13  vid = cv2.VideoCapture(0)
14
15  frame_width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
16  frame_height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
17
18  fourcc = cv2.VideoWriter_fourcc(*'mp4v')
19  out = cv2.VideoWriter('output.mp4', fourcc, 20.0, (frame_width, frame_height))
20
21  #Définir l'intervall de couleur du jaune
22  def get_limits(color):
23
24      c = np.uint8([[color]]) #couleurs du bg
25      hsvC = cv2.cvtColor(c, cv2.COLOR_BGR2HSV)
26
27      lowerLimit = hsvC[0][0][0] - 10, 100, 100
28      upperLimit = hsvC[0][0][0] + 10, 255, 255
29
30      lowerLimit = np.array(lowerLimit, dtype= np.uint8)
31      upperLimit = np.array(upperLimit, dtype= np.uint8)
32
33      return lowerLimit, upperLimit
34
35
```

```

34
35
36 while True :
37     RET , F = vid.read()
38     out.write(F)
39     hsvI = cv2.cvtColor(F, cv2.COLOR_BGR2HSV) # Convertir le BGR en espace colorimétrique HSV
40     lowerLimit, upperLimit = get_limits(color = yellow)
41
42     #trouver et defenir les valeurs de l'image_interval
43     mask = cv2.inRange(hsvI, lowerLimit, upperLimit )
44     mask_ = Image.fromarray(mask)
45     bbox = mask_.getbbox() #bouncing box
46
47     if bbox is not None :
48         x1, y1, x2, y2 = bbox
49         #gauche en haut, droite, couleur, epaisseur
50         F = cv2.rectangle(F,(x1, y1),(x2, y2),(0,255,0), 5)
51
52     cv2.imshow ("Direct", F)
53
54     KEY = cv2.waitKey(1) #Attend la touche pressée
55     if KEY == ord("q"):
56         break
57
58 vid.release()
59 out.release() #S'assurer que toutes les frames ont été écrites et que le fichier de sortie est
60 cv2.destroyAllWindows() # Detruit toutes les fenetres
61
62

```



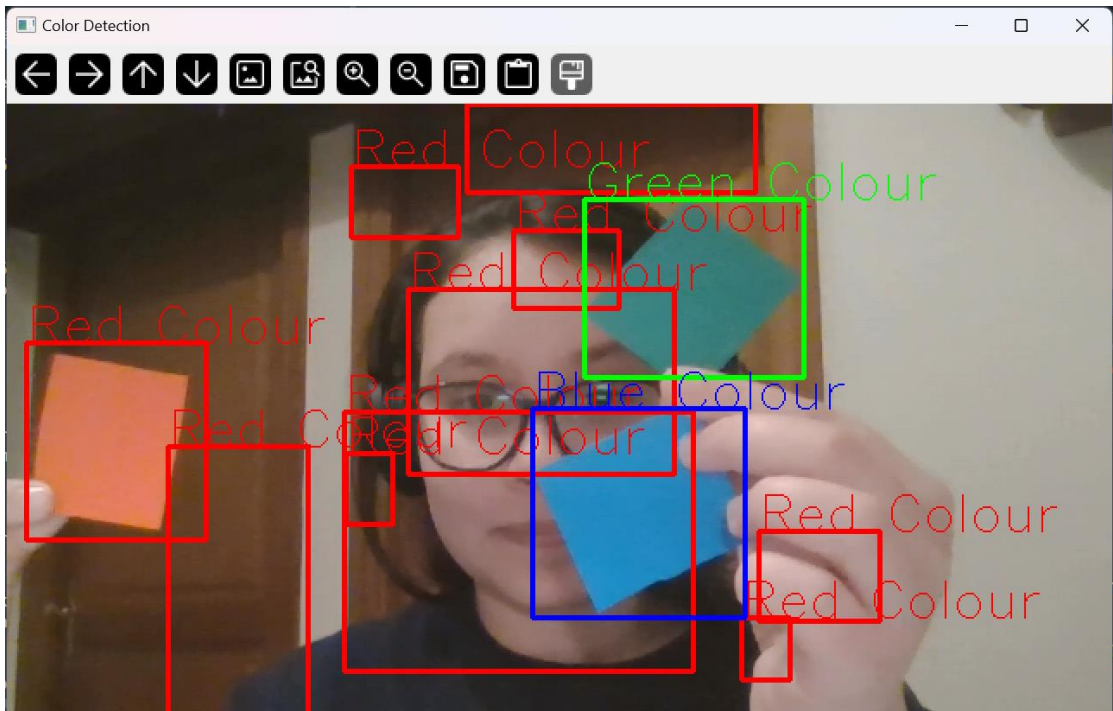
#### 4 - Premier essai de détection de trois couleurs :

```

1  #-*- coding: utf-8 -*-
2  ...
3  Created on Wed Jan 22 18:03:05 2025]
4
5  @author: Eleve
6  ...
7
8  ...
9  Détection de 3 couleurs en utilisant OpenCv + récupérer le flux vidéo (programme peut être i
10 ...
11
12 import numpy as np #appeler la bibliothèque numpy
13 import cv2
14
15 webcam = cv2.VideoCapture(0)
16
17 #capture des dimensions de la vidéo
18 #permettre la récupération de la largeur et de la hauteur de la vidéo capturée (webcam)
19 #ces valeurs sont ensuite stockées dans frame_width et frame_height.
20 frame_width = int(webcam.get(cv2.CAP_PROP_FRAME_WIDTH))
21 frame_height = int(webcam.get(cv2.CAP_PROP_FRAME_HEIGHT))
22
23 #spécifie le codec utilisé pour l'encodage de la vidéo en format .mp4. 'mp4v' est le codec i
24 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
25
26 #initialise l'objet cv2.VideoWriter, qui va écrire les frames dans le fichier output.mp4.
27 # (frame_width, frame_height) : définit la taille des images de la vidéo, la même taille q
28 out = cv2.VideoWriter('output.mp4', fourcc, 20.0, (frame_width, frame_height))
29
30 while (1):
31     ret, imageFrame = webcam.read()
32     # Convertir le BGR en espace colorimétrique HSV
33     hsvFrame = cv2.cvtColor(imageFrame, cv2.COLOR_BGR2HSV)
34
35     #Définir l'intervall de couleur du rouge
36     red_lower = np.array([0, 70, 50], np.uint8)
37     red_upper = np.array([10, 255, 255], np.uint8)
38
39     #Définir le mask
40     red_mask = cv2.inRange(hsvFrame, red_lower, red_upper)
41
42     #Définir l'intervall de couleur du vert
43     green_lower = np.array([40, 50, 50], np.uint8)
44     green_upper = np.array([90, 255, 255], np.uint8)
45     green_mask = cv2.inRange(hsvFrame, green_lower, green_upper)
46
47     #Définir l'intervall de couleur du bleu
48     blue_lower = np.array([100, 80, 2], np.uint8)
49     blue_upper = np.array([120, 255, 255], np.uint8)
50     blue_mask = cv2.inRange(hsvFrame, blue_lower, blue_upper)
51
52
53
54
55
56
57
58
59
60
61 red_mask = cv2.dilate(red_mask, kernel)
62 res_red = cv2.bitwise_and(imageFrame, imageFrame, mask=res_red)
63
64
65 #Mask vert
66 green_mask = cv2.dilate(green_mask, kernel)
67 res_green = cv2.bitwise_and(imageFrame, imageFrame, mask=green_mask)
68
69 #Mask bleu
70 blue_mask = cv2.dilate(blue_mask, kernel)
71 res_blue = cv2.bitwise_and(imageFrame, imageFrame, mask=blue_mask)
72
73 #Contour pour suivre la couleur rouge
74 contours, hierarchy = cv2.findContours(res_red, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
75
76 #Itère sur tous les contours trouvés dans l'image
77 for pic, contour in enumerate(contours): #permet d'obtenir l'index pic et le contour
78     area = cv2.contourArea(contour) #calculé la superficie du contour en pixel
79     if (area > 300): #condition si l'aire du contour est supérieure à 300 pixel
80         x, y, w, h = cv2.boundingRect(contour)#trouver un rectangle autour du contour e
81
82         imageFrame = cv2.rectangle(imageFrame, (x, y),(x + w, y + h),(0, 0, 255), 2)
83
84         cv2.putText(imageFrame, "Red Colour", (x, y),cv2.FONT_HERSHEY_SIMPLEX, 1.0,(0, 0,
85
86 #Contour pour suivre la couleur verte (même technique que pour le rouge)
87 contours, hierarchy = cv2.findContours(res_green, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
88
89 for pic, contour in enumerate(contours):
90     area = cv2.contourArea(contour)
91     if (area > 300):
92         x, y, w, h = cv2.boundingRect(contour)
93         imageFrame = cv2.rectangle(imageFrame, (x, y), (x + w, y + h),(0, 255, 0), 2)
94         cv2.putText(imageFrame, "Green Colour", (x, y),cv2.FONT_HERSHEY_SIMPLEX,1.0, (0
95
96 #Contour pour suivre la couleur bleu (même technique que pour le rouge)
97 contours, hierarchy = cv2.findContours(res_blue, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
98 for pic, contour in enumerate(contours):
99     area = cv2.contourArea(contour)
100    if (area > 300):
101        x, y, w, h = cv2.boundingRect(contour)
102        imageFrame = cv2.rectangle(imageFrame, (x, y),(x + w, y + h),(255, 0, 0), 2)
103        cv2.putText(imageFrame, "Blue Colour", (x, y),cv2.FONT_HERSHEY_SIMPLEX, 1.0, (2
104
105
106
107 cv2.imshow("Color Detection", imageFrame)
108 if cv2.waitKey(10) & 0xFF == ord('q'):
109     break
110
111 webcam.release()
112 out.release() #S'assurer que toutes les frames ont été écrites et que le fichier de sortie i
113 cv2.destroyAllWindows()
114

```

Les résultats sont encore à affiner :



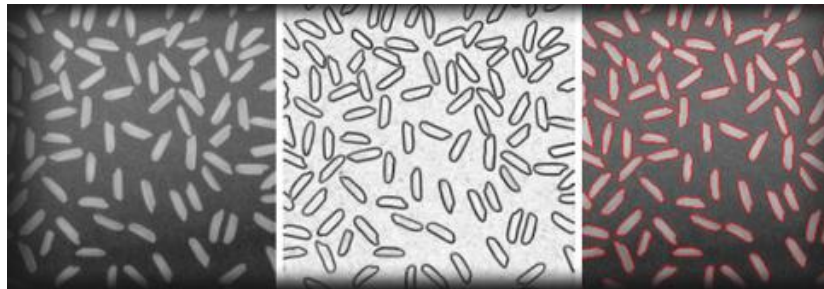
### 5 - Technique pour détecter un objet

La segmentation d'image est un processus crucial en vision par ordinateur qui consiste à diviser une image en plusieurs segments ou régions. Cette technique aide à simplifier la représentation d'une image, facilitant ainsi son analyse. OpenCV propose diverses méthodes pour la segmentation d'image. Nous allons juste présenter les grands principes efficaces pour

la segmentation d'image qu'utilise OpenCV, sans entrer dans les détails mathématiques.

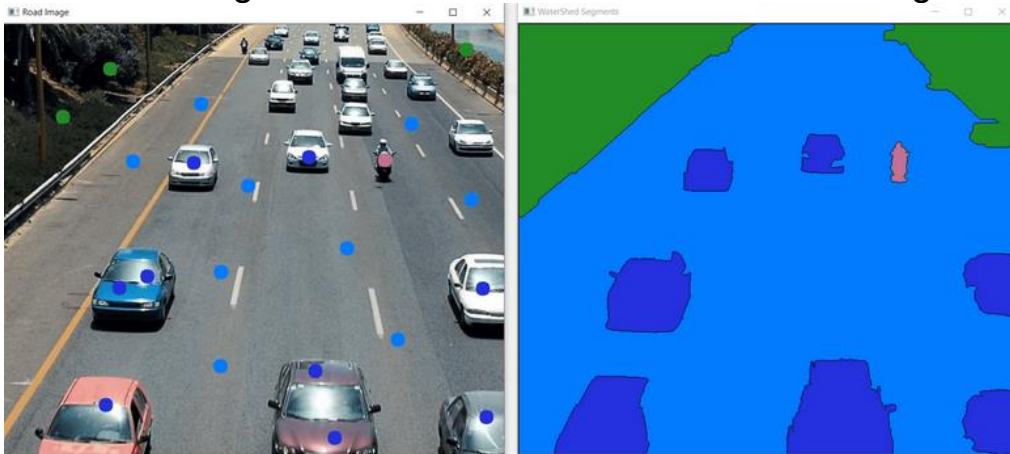
a) - Le seuillage est l'une des techniques les plus simples et les plus couramment utilisées pour la segmentation d'image. Il convertit une image en niveaux de gris en une image binaire, où les pixels sont classés comme avant-plan ou arrière-plan en fonction d'une valeur seuil.

b) - La détection de contours est une autre technique efficace pour la segmentation d'image. Elle identifie les limites des objets dans une image, permettant une segmentation précise.



c) - Un algorithme de segmentation avancé comme GrabCut utilise des coupes de graphe pour séparer l'avant-plan de l'arrière-plan. Il nécessite un rectangle englobant autour de l'objet d'intérêt.

La liste des algorithmes et des méthodes est assez longue.



## 6 - Premier essai pour la détection d'objet : un passage piéton

Le programme ci-après permet de détecter les formes caractéristiques des contours blanc des passages piétons.

```

1  '''
2  Détecter un passage piéton (image)
3  '''
4  import cv2
5  import numpy as np
6
7  #lire l'image zebra_lane.jpg
8  img = cv2.imread("zebra_lanes2.jpg")
9
10 #Traitement de l'image
11 #Converir l'image img de l'espace de couleur BGR (bleu, vert, rouge) vers un espace de couleur niveau de gris.
12 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
13
14 #Application d'un flou
15 blurred = cv2.GaussianBlur(gray, (15, 15), 6)
16
17 #Effectuer une seuilisation de l'image floutée pour la convertir en une image binaire (noir et blanc)
18 ret, thresh = cv2.threshold(blurred,180, 255,cv2.THRESH_BINARY)
19
20 #Détecer les contours dans l'image binaire
21 contours, hier = cv2.findContours(thresh.copy(),cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
22
23
24
25 for c in contours:
26     # Ignorer le contour si il n'est pas suffisamment large
27     if cv2.contourArea(c) < 2000:
28         continue
29
30     # Obtenir la plus petite superficie du rectangle possible
31     rect = cv2.minAreaRect(c)
32     box = cv2.boxPoints(rect)
33
34     #Convertir toutes les coordonnées float (décimales) en int (entiers)
35     box = np.int0(box)
36
37
38     #dessiner un rectangle rouge autour d'un objet (en utilisant les points du rectangle obtenus avec cv2.
39     #minAreaRect()) et pour enregistrer et afficher l'image avec ce rectangle dessiné
40     cv2.drawContours(img, [box], 0, (0, 0, 255), 2)
41     cv2.imwrite('zebra_lane5.jpg', img)
42     cv2.imshow("contours", img)
43
44 while True:
45     key = cv2.waitKey(1)
46     if key == 27:
47         break
48
49 cv2.destroyAllWindows()

```

Nous avons indiqué ci-dessous deux exemples d'images contenant un passage pour piéton, ainsi les résultats obtenus :

*Avant analyse de l'image :*



*Image après analyse :*



Nous constatons qu'en utilisant la bibliothèque OpenCV, il est possible d'obtenir un ensemble de résultats. Aussi bien au niveau de la reconnaissance des couleurs qu'au niveau de la détection d'objets particuliers, comme l'ont montré nos essais pour détecter un passage piéton.

Mais nous ne pouvons ignorer le nombre conséquent d'erreurs !

Au niveau des couleurs, la détermination précise des zones de couleurs n'est pas évidente. Si la zone de couleur détectée est trop restreinte, celle-ci ne sera pas détectée. Et si nous élargissons cette zone alors trop de zones de cette couleur seront détectées. Comme précédemment la couleur "rouge", qui est perçue à tellement d'endroit de l'image, que cela devient inexploitable.

Concernant la détection des passages piéton, soit nous détectons avec ce programme une partie du passage piéton, soit pas du tout et le passage pour piéton est perçu comme le rond-point.

Nous avons alors cherché un peu d'aide extérieur, et nous avons réussi à prendre contact avec Judy, une doctorante dans sa deuxième année à l'université de Lorraine. Judy a un diplôme d'informatique, et travaille durant sa thèse sur les thèmes de la robotique, de l'informatique et de l'IA : des sujets qui ont un lien direct avec notre projet.



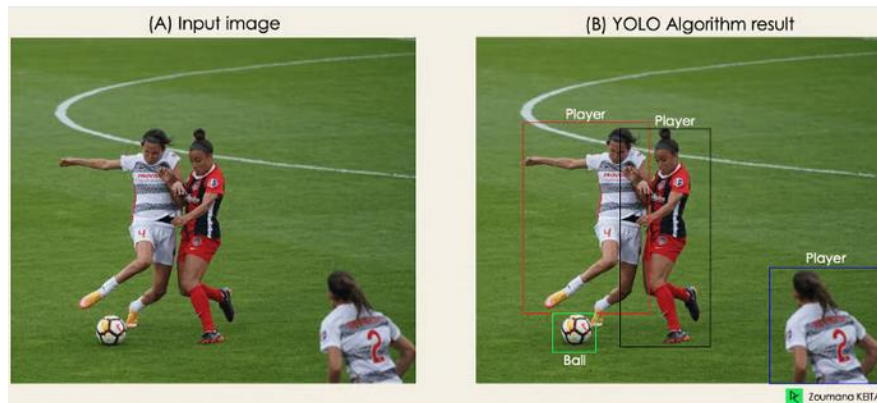
C'est notamment elle qui, après notre premier contact en visio-conférence, nous a orienté sur le modèle Ultralytics YOLO.

C'est un modèle de détection d'objets et de segmentation d'images en temps réel qui s'appuie sur des avancées de pointe en matière d'apprentissage profond et de vision par ordinateur. Et cela offre alors des performances inégalées en termes de rapidité et de précision par rapport à d'autres modèles comme la bibliothèque OpenCV. Nous nous sommes alors lancés sur cette nouvelle piste avec passion.

# Le modèle Ultralytics YOLO

## 1 - Détection d'objet avec YOLO.

Maintenant nous allons prendre comme exemple une application YOLO qui détecte les joueurs et les ballons de football à partir d'une image donnée.



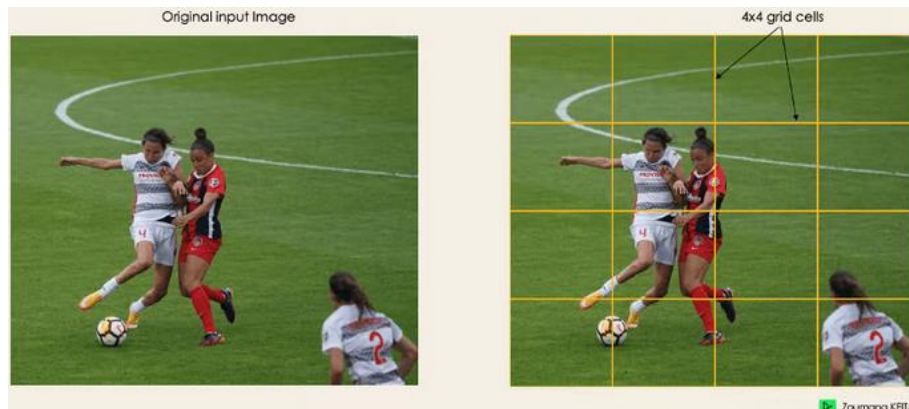
<https://www.datacamp.com/fr/blog/yolo-object-detection-explained>

L'algorithme fonctionne selon les quatre approches suivantes :

- blocs résiduels
- régression par boîte englobante
- intersection Over Unions ou IOU en abrégé
- suppression non maximale

### a) - Blocs résiduels :

Cette première étape commence par la division de l'image originale (A) en NxN cellules de grille de forme égale, où N, dans notre cas, est 4, comme le montre l'image de droite. Chaque cellule de la grille est chargée de localiser et de prédire la classe de l'objet qu'elle recouvre, ainsi que la valeur de probabilité/confiance.



## b) - Régression par boîte englobante :

L'étape suivante consiste à déterminer les boîtes de délimitation correspondant aux rectangles, en mettant en évidence tous les objets de l'image. Il peut y avoir autant de boîtes englobantes qu'il y a d'objets dans une image donnée.

YOLO détermine les attributs de ces boîtes de délimitation à l'aide d'un seul module de régression dans le format suivant, où Y est la représentation vectorielle finale de chaque boîte de délimitation.

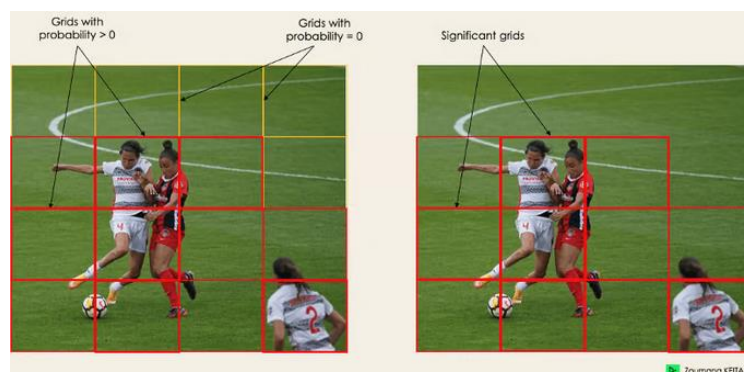
$Y = [pc, bx, by, bh, bw, c1, c2]$

pc correspond au score de probabilité de la grille contenant un objet.

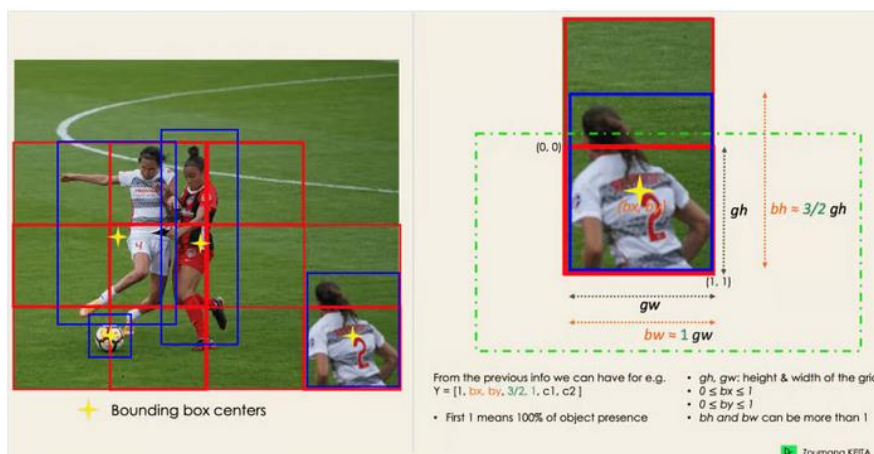
bx, by sont les coordonnées x et y du centre de la boîte englobante par rapport à la cellule de la grille enveloppante.

bhbw correspondent à la hauteur et à la largeur de la boîte de délimitation par rapport à la maille enveloppante.

c1 et c2 correspondent aux deux classes Joueur et Balle. Nous pouvons avoir autant de classes que votre cas d'utilisation l'exige.



Regardons de plus près le joueur en bas à droite :

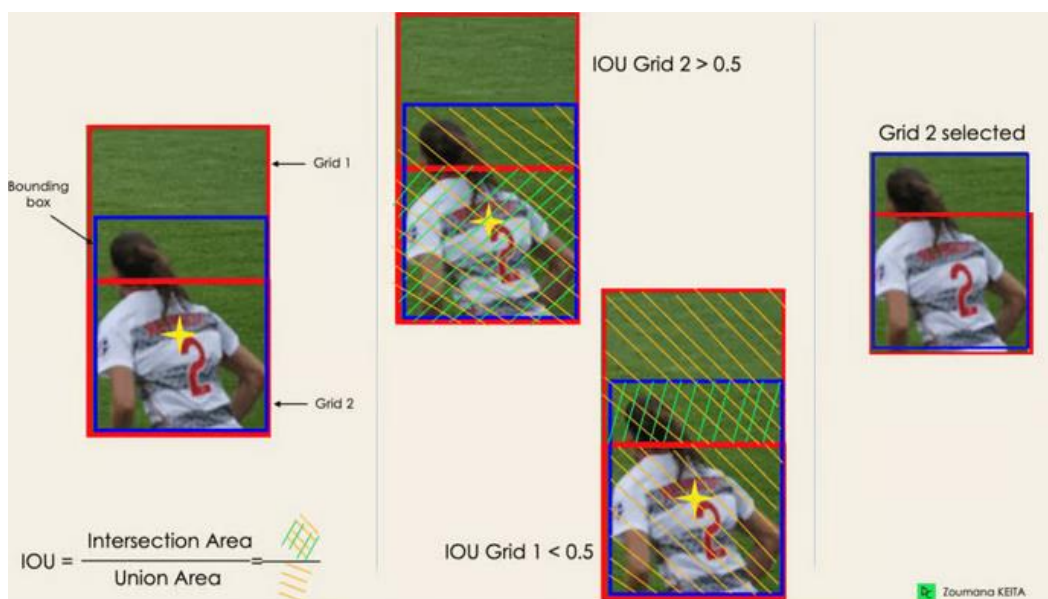


### c) - Intersection Over Unions ou IOU en abrégé :

La plupart du temps, un seul objet dans une image peut avoir plusieurs boîtes à grille candidates à la prédiction, même si toutes ne sont pas pertinentes. L'objectif de l'IOU (une valeur entre 0 et 1) est d'écarter ces cases de la grille pour ne garder que celles qui sont pertinentes. En voici la logique :

- L'utilisateur définit son seuil de sélection des reconnaissances de dettes, qui peut être, par exemple, de 0,5.
- Ensuite, YOLO calcule l'IOU de chaque cellule de la grille, qui correspond à la zone d'intersection divisée par la zone d'union.
- Enfin, il ignore la prédiction des cellules de la grille ayant un IOU  $\leq$  seuil et prend en compte celles ayant un IOU  $>$  seuil.

Nous avons ci-dessous une illustration de l'application du processus de sélection de la grille à l'objet situé en bas à gauche. Nous pouvons observer que l'objet avait à l'origine deux candidats à la grille, et que seule la "grille 2" a été sélectionnée à la fin.



### d) - suppression non maximale.

La fixation d'un seuil pour la reconnaissance n'est pas toujours suffisante, car un objet peut comporter plusieurs cases avec une reconnaissance supérieure au seuil, et le fait de laisser toutes ces cases peut inclure du bruit. C'est ici que nous pouvons utiliser la suppression non maximale pour ne conserver que les boîtes dont la probabilité de détection est la plus élevée.

## 1 - Premier contact avec YOLO et les images

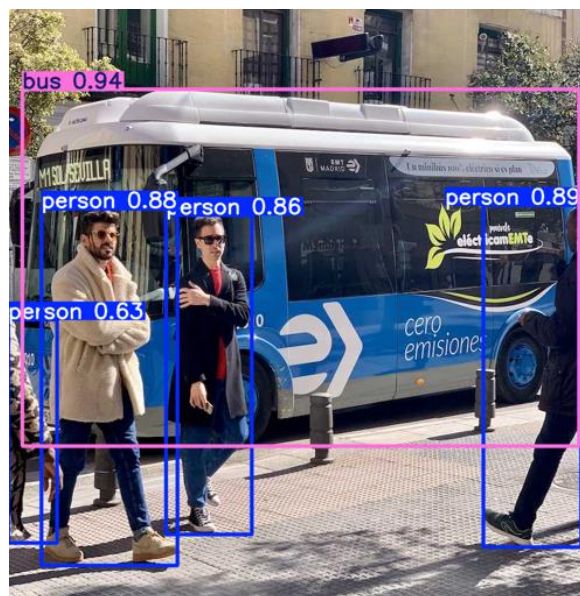
Nous avons réussi à implanter ce modèle sous Windows en passant par Anaconda. Cependant cela pose encore un problème sous Linux. Donc, sous Windows, avec l'intégrateur Anaconda, nous avons bien tous nos différents outils qui sont exploitables. Le programme ci-dessous utilise le modèle YOLO pour extraire des objets de l'image :

```
1 # -*- coding: utf-8 -*-
2 """
3 Test de détection d'objets YOLO
4 """
5
6 from ultralytics import YOLO #appeler la bibliothèque
7
8 # Chargement d'un modèle YOLO11n entraîné par COCO
9 model = YOLO("yolo11n.pt")
10
11 # Entraîne le modèle sur l'ensemble de données de l'exemple COCO8 pendant 5 époques
12 # (Un passage complet sur l'ensemble des données de formation)
13 results = model.train(data="coco8.yaml", epochs=1, imgsz=640)
14
15 # Exécuter l'inférence avec le modèle YOLO11n sur l'image 'bus.jpg'
16
17 results = model("https://ultralytics.com/images/bus.jpg")
18
19 #Test...
20 #results = model("E://Concours_Cgénial/Images_Feux_tricolores/image1.jpg")
21
22 #Voir les résultats
23 for result in results:
24     result.show()
25
26
```

Image avant analyse :



Image après analyse :



Nous constatons très vite la puissance de ce modèle. Il possède une bibliothèque d'objets de façon native, et celle-ci peut être étendue. De plus, il indique un degré de pertinence pour chaque objet détecté. Il nous reste à faire apprendre à ce modèle la reconnaissance des objets que l'on désire : les feux de signalisation et les passages piétons.

Mais quand est-il pour le traitement d'un flux vidéo ?

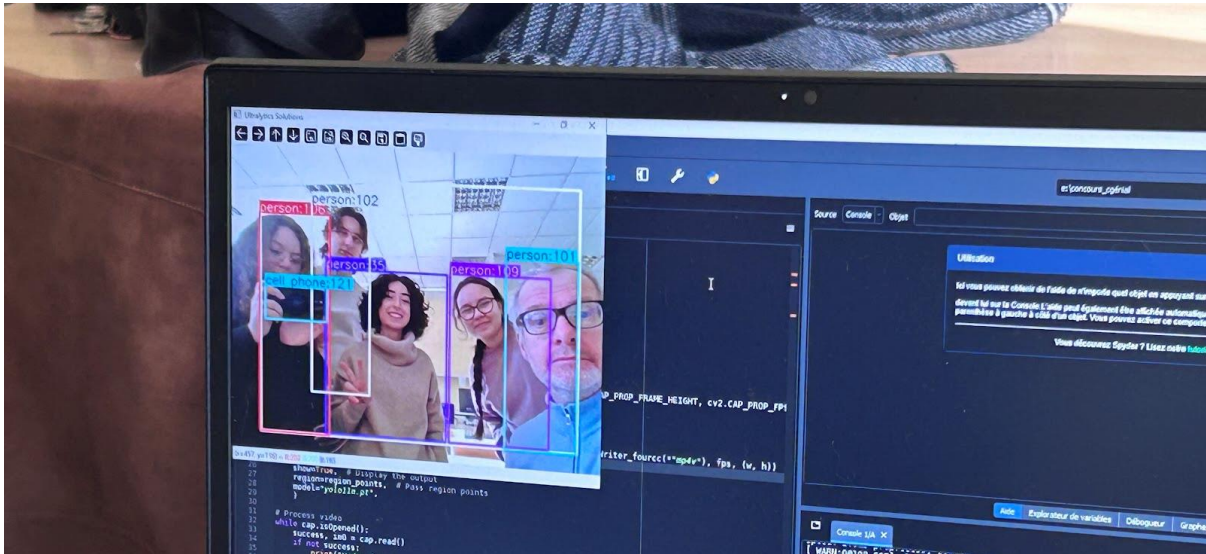
## 2 - Deuxième approche avec cette fois un flux vidéo

Pour exploiter au mieux le flux vidéo, nous allons utiliser aussi bien la bibliothèque OpenCV que le modèle YOLO :

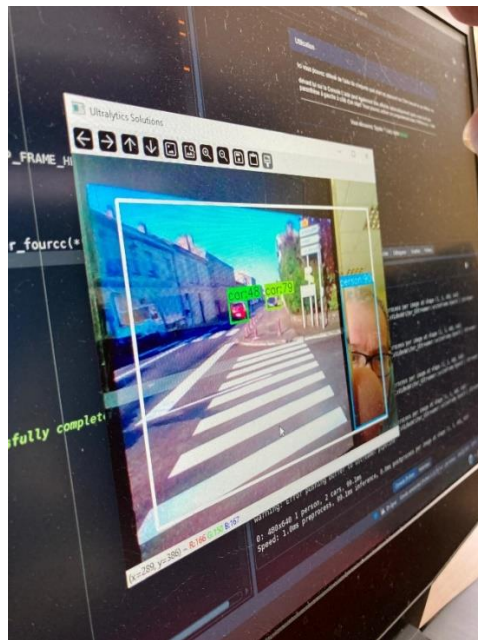
```
1 |
2 | import cv2
3 | import time
4 | import numpy as np
5 | import cv2 as cv
6 |
7 | from sys import exit
8 | CODE_TOUCHE_ECHAP = 27
9 |
10 | from ultralytics import solutions
11 |
12 | # Mettre 0 au lieu de 1 pour la caméra intégré du PC portable
13 | cap = cv2.VideoCapture(0)
14 |
15 | assert cap.isOpened(), "Error reading video file"
16 | w, h, fps = (int(cap.get(x)) for x in (cv2.CAP_PROP_FRAME_WIDTH, cv2.CAP_PROP_FRAME_HEIGHT, cv2.CAP_PROP_FPS))
17 |
18 | # Define region points
19 | region_points = [(50, 50), (50, 450), (600, 50), (600, 450)]
20 |
21 | # Video writer
22 | video_writer = cv2.VideoWriter("object_counting_output.avi", cv2.VideoWriter_fourcc("mp4v"), fps, (w, h))
23 |
24 | # Init TrackZone (Object Tracking in Zones, not complete frame)
25 | trackzone = solutions.TrackZone(
26 |     show=True, # Display the output
27 |     region=region_points, # Pass region points
28 |     model="yololln.pt",
29 | )
30 |
31 | # Process video
32 | while cap.isOpened():
33 |     success, im0 = cap.read()
34 |     if not success:
35 |         print("Video frame is empty or video processing has been successfully completed.")
36 |         break
37 |     im0 = trackzone.trackzone(im0)
38 |     video_writer.write(im0)
39 |
40 |     code_touche_clavier = cv.waitKey(20)
41 |     if code_touche_clavier == CODE_TOUCHE_ECHAP:
42 |         break
43 |
44 | cap.release()
45 | video_writer.release()
46 | cv2.destroyAllWindows()
47 |
48 |
```

Dans un premier temps, nous avons fait des essais pour réaliser le même type de détection que sur une simple image. Mais, ci-dessous, la capture d'écran était sur le flux vidéo de la caméra de l'ordinateur. Nous avons été bluffés par la puissance et la rapidité de l'analyse réalisée, même si ce

n'était que la détection d'individus (notre équipe), et pas encore les objets voulus.



Puis nous avons commencé les premiers tests avec la capture vidéo d'un environnement extérieur. Nous pouvons constater que le passage piéton n'est pas encore identifié. En effet, les voitures sont bien repérées, mais le passage piéton, qui n'est obstrué par aucun obstacle, et qui est bien en vue, n'est absolument pas repéré.



## Design et support du dispositif

Notre première idée a été de fixer la caméra directement sur des lunettes. Cependant, d'un point de vue logistique, nous avons dû repenser le design de notre projet puisque la taille de la caméra empêchait un montage efficace et durable. En effet, la plupart des caméras USB ont un conducteur électrique situé à l'arrière de la caméra. Par conséquent, qu'il

s'agisse de la taille, de la forme ou du poids de la caméra, rien ne permettait de l'associer au design fin et élégant de lunettes.



Viens donc une des premières étapes de notre questionnement : le support.

*Quel support utiliser pour assurer la prise d'un maximum d'informations tout en restant utile peu encombrant pour l'utilisateur ?*

Notre attention s'est tout d'abord portée sur l'idée d'un bracelet ou d'un système se basant sur le Bluetooth. Cependant nous savions que certaines personnes malvoyantes ou aveugles utilisent déjà des systèmes d'aides ou de directions (tel qu'un GPS) et nécessite donc l'utilisation d'écouteurs.

Ne pouvant donc pas perturber la communication filaire ou Bluetooth nous avons trouvé l'idée du bandeau la plus judicieuse. En effet comme mentionné précédemment le bandeau permet de rediriger le système filaire connectant la caméra au boîtier le long du dos de la personne sans gêner ses mouvements.

Viens donc ensuite la question du boîtier, trop volumineux pour simplement rester dans une poche.



Nous avons donc suggéré l'idée d'une ceinture protectrice ou serait attaché le boîtier. L'emplacement du boîtier reste encore à être évalué puisqu'il ne doit pas gêner le déplacement de la personne et ne doit donc pas l'empêcher de s'asseoir. Il s'agirait donc d'étudier laquelle de nos deux options serait la plus intéressante : le boîtier situé dans le dos de la personne autour d'une ceinture assez haute pour ne pas interférer avec la capacité à s'asseoir ou un boîtier situé sur le côté à condition qu'il ne soit pas protubérant et assez protégé. La ceinture devrait idéalement être imperméable et recouvrir le boîtier. Elle agirait donc comme une poche protectrice.

Ce système peut se porter sous ou sur des vêtements et se voudrait le plus discret et le moins encombrant possible.

## **Partenariat**

Nous avons à cœur de faire un partenariat avec une association pour personnes malvoyantes et aveugles. Le cœur même de notre projet étant le bien-être et l'intégration de ces personnes atteintes de déficience visuelle, nous voulions avoir l'avis et les conseils des personnes concernées. Dès le début de notre projet, nous avons donc contacté plusieurs fois l'association Voir Ensemble, localisée en Moselle. Nous n'avons malheureusement pas eu de réponse, mais peut-être que prochainement nous pourrions leur présenter notre projet et possiblement l'affiner selon les retours que nous aurons.

## **Evolution du projet.**

Ce projet est actuellement en constante évolution. Tant le niveau de technicité requis est important pour le traitement du flux vidéo. A SUIVRE!

# Conclusion

Pour conclure, à l'aide de ce travail d'équipe, nous pouvons dire qu'après avoir fait face à de nombreux défis et avoir effectué plusieurs changements imprévus, mais nécessaire, nous avons réussi à avoir les résultats attendus. Ce projet nous a permis d'en apprendre plus sur le travail d'équipe, la recherche, l'informatique, l'organisation et la concrétisation d'idées ! Notre programme arrive à repérer, avec une marge d'erreur raisonnable, les passages piétons et les feux de signalisation, ce qui pourrait donc assurer la sécurité des personnes malvoyantes ou aveugles au sein des villes. Nous avons encore bien sûr des détails à améliorer, d'un point de vue concret par rapport à la taille de la caméra sur le bandeau, ou de la luminosité de l'environnement... Mais avec une identification fiable et l'avancée des outils technologiques, nous aimerions la concrétisation de ce projet : un outil qui vient compléter les systèmes déjà existants, en offrant une sécurité supplémentaire, qui pourrait être utilisé durablement !

## Remerciements

Mr. Pierre, professeur du Club Robotique qui nous a aidé dans toutes nos initiatives.

Mr. Nivoix, professeur de NSI de notre lycée qui n'a pas hésité à nous guider dans notre projet.

Judy AKL, doctorante de l'université de Lorraine qui a pris de son temps pour nous guider dans nos idées.

## Sources

[https://nsimichelet91.github.io/snt/Theme1\\_Image\\_numerique/cours/](https://nsimichelet91.github.io/snt/Theme1_Image_numerique/cours/)

<https://www.datacamp.com/fr/tutorial/installing-anaconda-windows>

<https://stacklima.com/configurer-opencv-avec-lenvironnement-anaconda/>

<https://learn.microsoft.com/fr-fr/windows/ai/windows-ml/tutorials/>

<https://www.anaconda.com/>

<https://opencv.org/>

<https://docs.ultralytics.com/fr>

“*Traitement d'images et de vidéos avec OpenCV 4 en Python*” de Laurent Berger Éditions D-Booker