X) Suivi Temps Réel d'un Ballon avec Programmation Python et Caméra JeVois

Cette section se concentre sur le développement d'un programme conçu pour le suivi en temps réel d'un objet spécifique. Pour cette étude, l'objet en question est un ballon jaune, sélectionné pour son utilisation dans la compétition 'Robotcup'. Le programme est capable de déterminer et de transmettre les coordonnées x et y du ballon. L'objectif principal est de permettre à une caméra JeVois de détecter et d'envoyer les positions successives du ballon à un ordinateur via le port série, à un instant précis.

A) Codage Python pour le Suivi sur Image

Il est important de disposer d'abord d'un programme Python fonctionnel pour bien comprendre le mécanisme de suivi d’objet avant de l'implémenter sur la caméra JeVois. Cela permet d'assurer le bon fonctionnement du système mais surtout de le comprendre. Pour structurer la résolution du problème, le processus commence par le suivi de l'objet sur une image unique. Cette étape est suivie par l'application du même principe sur une vidéo, qui est essentiellement une séquence d'images. Ainsi, en appliquant le programme de suivi à chaque image de la vidéo, on peut réaliser le suivi en continu.

Voici le résultat final désiré :

Figure 1 : Suivi d’un ballon sur une image

Pour commencer, on ajoute deux librairies : OpenCV et NumPy. OpenCV permet de travailler avec les images, aidant à repérer et suivre les objets. NumPy est utile pour les opérations numériques, notamment dans le traitement des images de vidéos, permettant de traiter chaque image pour le suivi des objets. Ensemble, OpenCV et NumPy sont essentielles pour détecter et suivre les objets, que ce soit dans des images ou des vidéos.

import cv2

import numpy as np

Après avoir importé l’image souhaitée à l’aide de la fonction cv2.imread(« Nom\_image.jpg »), il faut maintenant filtrer l’image.

imagehsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

H, S, V = cv2.split(imagehsv)

La première ligne du code convertit le mode de couleur de l'image, en choisissant entre HSV ou RGB, par exemple. Puis la seconde ligne permet de diviser l'image en trois composantes distinctes : la teinte (Hue), la saturation (Saturation), et la luminosité (Value). Cette séparation est essentielle pour isoler certaines couleurs dans l'image, comme le jaune, dans le cas présent.

Ensuite, pour appliquer le filtre de couleur, il faut fixer des limites précises. Ces seuils sont établis après expérimentation et ajustements pour garantir leur efficacité dans différentes conditions. Cela explique donc l’utilité de upper\_yellow et lower\_yellow dans les lignes suivantes :

#Filtre Jaune

lower\_yellow = np.array([20, 100, 100])

upper\_yellow = np.array([30,255,255])

imagemaskyellow = cv2.inRange(imagehsv, lower\_yellow, upper\_yellow)

yellow\_filtered\_img = cv2.bitwise\_and(img, img, mask=imagemaskyellow)

La fonction np.array permet donc de définir une teinte HSV précise. Ensuite, un masque est créé avec les limites de jaunes définies auparavant (cv2.inRange). Pour finir, l’image jaune final est créé en ajoutant à l’image initiale le masque créé à l’aide de la fonction cv2.bitwise\_and. Ce qui donne l’image ci-dessous :

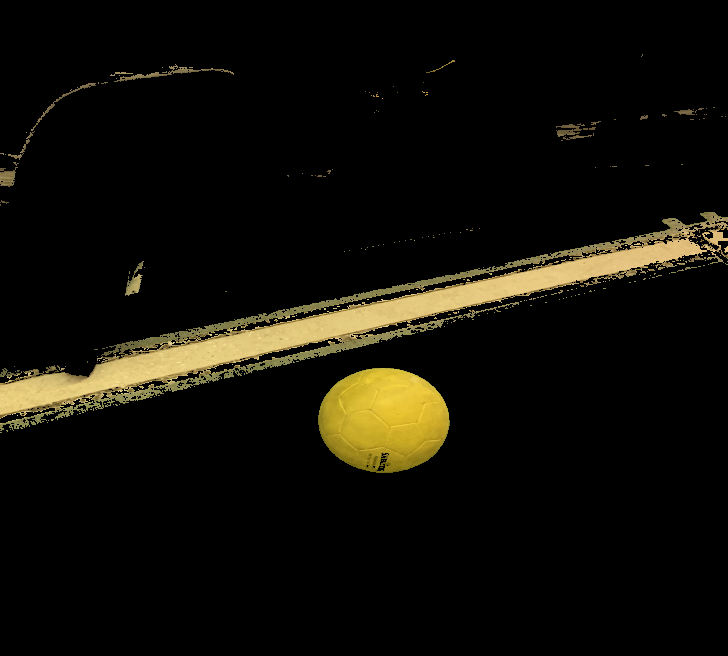


Figure 2 : Filtre jaune de l’image

Le programme fonctionne donc bien, l’image est filtrée par le jaune. Néanmoins plusieurs artéfacts de décors « polluent » le filtre. En effet il est normal que dans le décor qu’il y est des parties jaunes. C’est pour cela qu’il est maintenant important de filtrer l’image par rapport au contour de l’objet. Si un objet est jaune et rond, alors on en déduit que c’est un ballon. Maintenant pour faire un contour sur le ballon comme sur l’image 1, il est nécessaire d’utiliser les blobs, fonction qui permet de détourer certain objet.

La fonction qui permet de trouver les contours de l’image est la fonction :

contours, \_ = cv2.findContours(gris, cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

Maintenant la variable valeur est un tableau qui contient tous les contours de l’image (dans le cas présent, ce sera tous les contours des objet jaunes). Or comme vu auparavant il est nécessaire de trier les contours du ballon. Il faut donc parcourir le tableau de contour et vérifier si les contours sont assez gros (pour éviter de contourer du bruit).

Ensuite il est nécessaire de vérifier si les contours sont circulaires. Sachant que la réalité n’est pas parfaite, le ballon n’est pas parfaitement rond et en plus le programme étale l’image, ce qui au final rend le ballon proche d’une ellipse. Il est donc important de sélectionner les contours qui sont des ellipse (dans le cas parfait, le cercle serait un cas particulier de l’ellipse).

((x, y), rayon) = cv2.minEnclosingCircle(contour)

        # Vérifier si le contour est suffisamment circulaire

        area = cv2.contourArea(contour)

        perimeter = cv2.arcLength(contour, True)

        circularity = 4 \* np.pi \* (area / (perimeter \* perimeter))

C’est donc cette partie qui permet de calculer la circularité de l’objet.

if 0.8 <= circularity <= 1.2:

cv2.circle(img, (int(x), int(y)), int(rayon), (0, 255, 0), 2)

Ensuite il suffit simplement prendre les contours qui sont presque circulaires. C’est seulement après avoir vérifié ceci que le programme trace le cercle autour de l’objet.

Ceci permet donc de n’avoir que le ballon à la fin et donc de pouvoir obtenir l’image finale (voir image 1)

B) Application python pour le suivi sur vidéo

La partie précédente a expliqué le processus de détection du ballon jaune sur une image statique. L'étape suivante consiste à étendre cette méthode à une vidéo. Comme indiqué précédemment, une vidéo se compose d'une série d'images (ou frames). Pour réaliser le suivi en vidéo, il est donc nécessaire d'exécuter le même programme sur chaque image de la séquence vidéo. Précédemment, pour télécharger l’image, c’était la fonction cv2.imread("Nom\_fichier.jpg"), mais maintenant pour télécharger la vidéo, ce sera la fonction cv2.VideoCapture('nom\_fichier.mp4').

Maintenant il est nécessaire d’implémenter la fonction cap.isOpened. Si la vidéo ou la caméra est accessible et prête à être lue ou capturée, cap.isOpened() retourne True. Si elle ne peut pas accéder à la vidéo ou si quelque chose ne va pas (par exemple, si le chemin du fichier est incorrect ou si la caméra n'est pas disponible), elle retourne False. C’est pour cela que cette fonction sera utilisée dans un while :

while cap.isOpened():

Cela permet donc de ne pas planter si la vidéo ou la caméra à un problème.

Ensuite, pour avoir chaque frame de la vidéo, ce sera la fonction :

ret, img = cap.read()

Ret est un booléen signalant le succès (True) ou l'échec (False) de la lecture d'une frame vidéo, tandis qu'img stocke cette frame. On utilise ret pour déterminer la fin de la vidéo. Ainsi, si ret est False, cela indique la fin du flux vidéo, et le programme quitte alors la boucle while :

if not ret:

        break

Maintenant, comme pour l’image statique, le programme filtre l’image et trace un cercle. Le programme traite donc chaque image de la vidéo est la traite pour au final avoir un suivi de l’objet.

Maintenant que le suivi d’objet fonctionne pour une vidéo préenregistré, il suffit de reporter le programme sur la camera JeVois, ce qui permettra de faire du suivi d’image en temps réel.