Xavier Jannin
Aubin Jousselin

Vision : TP 1

Couleur

Compte Rendu

CPE Lyon – 5ETI 2019-2020

L'objectif de ce TP est d'observer et tester différentes méthodes pour permettre l'identification d'objets contenus dans une image à partir de leur couleur.

Question 1 : Test du code `balls_tracker.py`

En lançant le script `balls_tracker.py`, une fenêtre s'ouvre et affiche une vidéo :

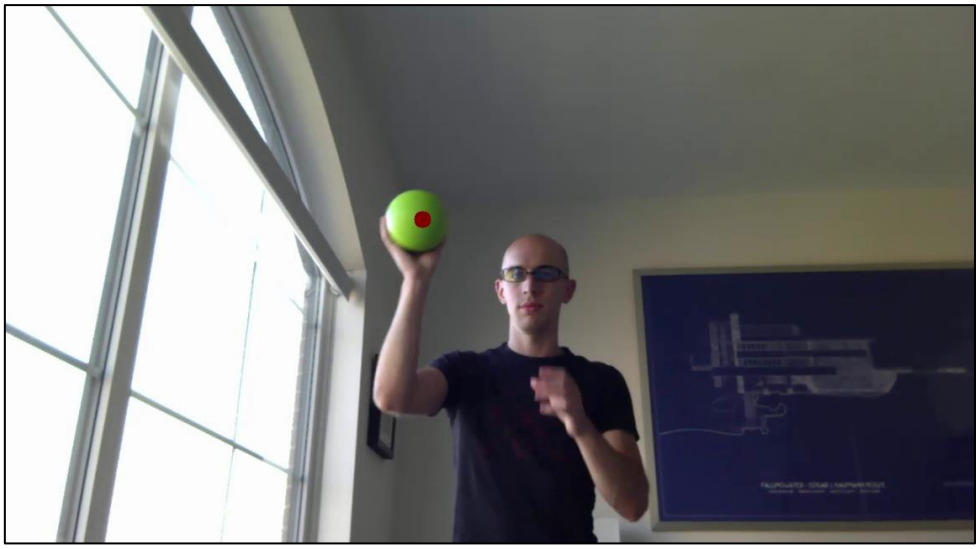


figure 1 – Exécution du programme `balls_tracker.py` sur la vidéo `ball.mp4`

Nous pouvons ainsi observer que ce code permet de trouver les éléments vert contenus dans une image, et de placer un point au centre de l'objet détecté. (Le point se trouve plus précisément au centre de l'ensemble des pixels détectés comme vert)

Question 2 : Détection de couleurs

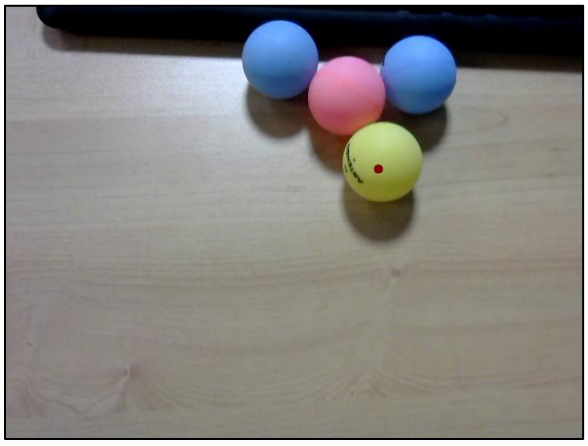
L'objectif est à présent de permettre la détection de balles de différentes couleurs. Ainsi, à partir du programme précédent, il suffit juste de déterminer les bornes permettant d'identifier une couleur, pour chaque couleur.

Nous avons décidé qu'il serait intéressant, au lieu de directement donner les valeurs des bornes, de générer les limites à partir d'une couleur BGR. Cela permet ainsi de plus facilement déterminer des bornes pour chaque couleur :

```python
# Get threshold for each color (blue, green, pink or yellow):
def getThreshold(color):

    if color == 'blue': bgr_color = [133, 82, 42]
    elif color == 'green': bgr_color = [135, 161, 99]
    elif color == 'pink': bgr_color = [86, 70, 167]
    elif color == 'yellow': bgr_color = [160, 219, 200]
    else:
      print("unknown color for threshold: default 'green'")
      bgr_color = [135, 161, 99]

    # Get HSV value from BGR color:
    bgr_color = np.uint8([[bgr_color]])
    hsv_color = cv2.cvtColor(bgr_color, cv2.COLOR_BGR2HSV)

    # Get Hue value:
    h_color = hsv_color[0][0][0]

    # Lower bound:
    if h_color < 20:
      h_color = 20

    # Threshold:
    lower_color = np.array([h_color - 20, 80, 80])
    upper_color = np.array([h_color + 20, 255, 255])

    return [lower_color, upper_color]
```

Nous avons ainsi pu correctement isoler les pixels contenant la couleur verte, jaune ou rose, qui permet ensuite d'en déduire le centre de chaque balle, pour la vidéo `ball3.mp4` :

Vert



Jaune

Rose

Cependant, la méthode que nous avons utilisée, n'apparaît pas très efficace pour isoler la balle de couleur bleue. En effet, le clavier présent dans l'image est aussi détecté comme faisant parti de la couleur bleue, ce qui signifie que le point obtenu ne correspond pas au centre de la balle :

Bleu

Pour corriger ce problème, nous avons simplement défini des bornes pour uniquement détecter la couleur bleue :

```
# Get threshold for each color (blue, green, pink or yellow):
def getThreshold(color):

    if color == 'blue':
        # Returns directly threshold:
        lower_blue = np.array([80, 140, 140])
        upper_blue = np.array([120, 255, 255])
        return [lower_blue, upper_blue]

    elif color == 'green': bgr_color = [135, 161, 99]
    ...
```

Ce qui permet d'obtenir la détection suivante, avec le centre correctement placé sur la balle bleu :

Bleu

Question 4 : Explication du programme `autopylot_agent.py`

Le programme `autopylot_agent.py` permet de générer les contrôles nécessaires à un drone (*zap*, *phi*, *theta*, *gaz*, *yaw*) pour qu'une balle verte se trouve centrée dans l'image reçue par la caméra.

Pour cela, le programme `greenball_tracker.py` est utilisé pour trouver le centre de l'objet vert présent devant la caméra.

Ensuite, un PID est utilisé pour générer les commandes qui permettrait de corriger l'écart entre le centre de l'objet détecté et le centre de l'image (calculé proportionnellement par rapport à l'image).

Ainsi, le drone n'essaye pas de se placer à une certaine distance de la balle, mais seulement d'être correctement face à elle.

Question 5 : Comportement avec deux boules de la même couleur

Problème :

Le programme `greenball_tracker.py` permet de récupérer le centre de tous les pixels verts contenu dans l'image. Ainsi, si une image contient 2 balles, le programme renverra un point situé au milieu de ces deux balles :

figure 2 – Exécution du programme `balls_tracker.py` sur la vidéo `ball2.mp4`

Ce comportement n'apparaît pas pertinent puisque le drone devrait être capable de suivre continuellement un élément sans qu'il soit dérangé par un autre objet.

Solution :

Le programme `track.py` propose une solution à ce problème :

Il détecte les contours de l'objet, grâce à la commande `cv2.findContours()`, ce qui permet de récupérer séparément des blocs de pixels. Ensuite, seul l'objet possédant la plus grande surface est choisi pour récupérer le centre de celui-ci.

Question 6 : Modification du code pour obtenir un comportement équivalent

Pour déterminer uniquement le centre d'un objet, sans qu'il soit gêné par un autre, il suffit d'utiliser la fonction `cv2.findContours()` pour séparer les blocs de pixels, puis de garder le bloc avec le plus de pixels.

Code ajouté pour obtenir le comportement équivalent :

```
def track(image):
    ...

    # Blur the mask
    bmask = cv2.GaussianBlur(mask, (5,5),0)

    # Find Contours
    contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    showingCNTs = [] # Contours that are visible
    areas = [] # The areas of the contours

    # Find Specific contours
    for cnt in contours:
        approx = cv2.approxPolyDP(cnt, 0.1 * cv2.arcLength(cnt, True), True)
        area = cv2.contourArea(cnt)
        if area > 300:
            areas.append(area)
            showingCNTs.append(cnt)

    # Only Highlight the largest object
    centroid_x, centroid_y = None, None
    if len(areas) > 0:
        maxIndex = areas.index(max(areas))
        cnt = showingCNTs[maxIndex]

        # Take the moments to get the centroid
        moments = cv2.moments(cnt)
        m00 = moments['m00']
        centroid_x, centroid_y = None, None
        if m00 != 0:
            centroid_x = int(moments['m10']/m00)
            centroid_y = int(moments['m01']/m00)

        # Other method to find centroid:
        #x, y, w, h = cv2.boundingRect(cnt)
        #centroid_x, centroid_y = x + w/2, y + h/2

    # Assume no centroid
    ctr = (-1, -1)

    # Use centroid if it exists
    if centroid_x != None and centroid_y != None:
    ...
```

Résultat :

figure 3 – Traitement sur la détection de la couleur verte pour localiser le centre de la balle

Cette méthode permet donc de bien isoler les pixels verts d'une image afin de récupérer le centre de l'objet détecté.

Question 7 : Détection des balles roses à partir du code précédent

Pour permettre la détection des balles sur la vidéo `ball4.mp4`, il suffit de reprendre le code précédent et de simplement modifier le threshold pour correctement détecter la couleur rose :

```
def track(image):

    '''Accepts BGR image as Numpy array
       Returns: (x,y) coordinates of centroid if found
                (-1,-1) if no centroid was found
                None if user hit ESC
    '''

    # Blur the image to reduce noise
    blur = cv2.GaussianBlur(image, (5,5),0)

    # Convert BGR to HSV
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

    # Threshold the HSV image for only pink colors
    lower_pink = np.array([140, 100, 100])
    upper_pink = np.array([179, 255, 255])

    # Threshold the HSV image to get only pink colors
    mask = cv2.inRange(hsv, lower_pink, upper_pink)

    ...
```

Ce code permet alors de bien trouver le centre d'une balle rose :

figure 4 – Traitement sur la détection de la couleur rose pour localiser le centre de la balle

Cependant, comme le programme cherche à afficher le centre de la balle possédant le plus de pixels contenus dans l'image, il arrive que le centre se téléporte d'une balle à l'autre :

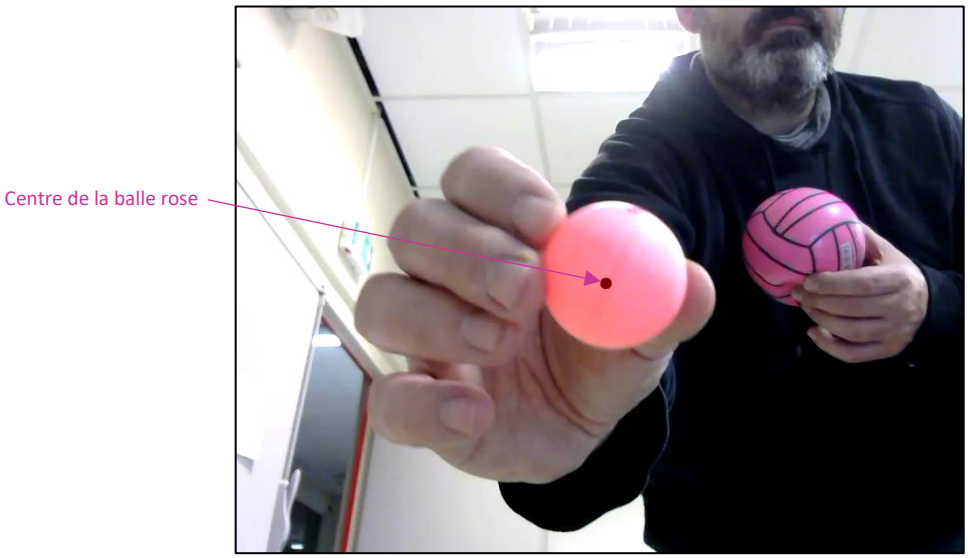


figure 5 – Observation de la téléportation d'une balle rose vers une autre

Question 8 : Proposition de correction de la téléportation du centre de la balle détecté

Pour garder le centre toujours sur la même balle, il est possible d'enregistrer la position du centre obtenue d'une image, puis de vérifier à l'image suivante que le centre trouvé est proche de celui de l'image précédente. Ainsi, cela permet de s'assurer de la continuité du déplacement du centre tout au long de la vidéo. Cependant, cette solution ne permettra pas de distinguer les balles si elles se superposent durant la vidéo.

(Cette solution nécessite aussi que le premier élément détecté soit bien une balle et non du bruit : nécessite alors quelques traitements supplémentaires pour détecter uniquement les éléments importants de l'image)

Question 9 : Application de la proposition

La mise en place de la solution reprend grandement le programme développé durant la question 7. Cependant, il enregistre la position du centre obtenu de l'image en cours pour l'image suivante. Ainsi, il est nécessaire de déterminer les centres de chaque contour détecté et de choisir celui dont la position est proche du centre précédent.

Voici le programme obtenu :

```
#!/usr/bin/env python
import cv2
import numpy as np

pctr = (-1, -1)


def track(image):
    '''Accepts BGR image as Numpy array
       Returns: (x,y) coordinates of centroid if found
                (-1,-1) if no centroid was found
                None if user hit ESC
    '''
    global pctr

    # Blur the image to reduce noise
    blur = cv2.GaussianBlur(image, (5, 5), 0)

    # Convert BGR to HSV
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

    # Threshold the HSV image for only pink colors
    lower_pink = np.array([140, 100, 100])
    upper_pink = np.array([179, 255, 255])

    # Threshold the HSV image to get only pink colors
    mask = cv2.inRange(hsv, lower_pink, upper_pink)

    # Image processing
    element = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
    bmask = cv2.erode(mask, element, iterations=3)
    element = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    bmask = cv2.dilate(bmask, element, iterations=20)

    # Find Contours
    #contours, hierarchy =
    #     cv2.findContours(bmask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    im, contours, hierarchy =
          cv2.findContours(bmask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    showingCNTs = []  # Contours that are visible
    areas = []  # The areas of the contours

    # Find Specific contours
    for cnt in contours:
        approx = cv2.approxPolyDP(cnt, 0.1*cv2.arcLength(cnt, True), True)
        area = cv2.contourArea(cnt)
        if area > 300:
            areas.append(area)
            showingCNTs.append(cnt)

    # Only Highlight the largest object
    centroid_x, centroid_y = None, None

    # Get the closest centroid to previous one:
->  for cnt in showingCNTs:

        # Take the moments to get the centroid
        x, y = getCentroid(cnt)

        # Compare from previous center, if it exists:
        d = 100
        if pctr[0] == -1 and pctr[1] == -1:
            centroid_x, centroid_y = x, y
        elif abs(x - pctr[0]) < d and abs(y - pctr[1]) < d:
            centroid_x, centroid_y = x, y

    ...
```

```
    ...

    # Assume no centroid
    ctr = (-1, -1)

    # Use centroid if it exists
    if centroid_x != None and centroid_y != None:
        ctr = (centroid_x, centroid_y)

        # Put black circle in at centroid in image
        cv2.circle(image, ctr, 10, (0, 0, 125), -1)

    # Save previous point:
    pctr = ctr

    # Display full-color image
    cv2.imshow('PinkBallTracker', image)

    # Force image display, setting centroid to None on ESC key input
    if cv2.waitKey(1) & 0xFF == 27:
        ctr = None

    # Return coordinates of centroid
    return ctr


# Get centroid from contour:
def getCentroid(cnt):
    moments = cv2.moments(cnt)
    m00 = moments['m00']
    centroid_x, centroid_y = None, None
    if m00 != 0:
        centroid_x = int(moments['m10']/m00)
        centroid_y = int(moments['m01']/m00)

    return centroid_x, centroid_y


if __name__ == '__main__':
    capture = cv2.VideoCapture('ball4.mp4')

    while True:
        okay, image = capture.read()
        if okay:
            if not track(image):
                break
            if cv2.waitKey(1) & 0xFF == 27:
                break
        else:
            print('Capture failed')
            break
```

Ce programme permet ainsi de bien suivre la balle sans que le centre se téléporte :

figure 6 – Suivi de la balle rose, sans téléportation du centre vers l'autre balle

Question 10 : Apprentissage de la couleur de l'objet se trouvant au milieu de l'image

Pour permettre l'apprentissage de la couleur d'un objet, il suffit de récupérer la valeur HSV du pixel au milieu de l'image afin de pouvoir ensuite créer un *threshold* qui permettra de détecter l'ensemble de l'objet.

Pour simplifier, nous avons décidé que ce serait le pixel au milieu de la première image d'une vidéo qui serait utilisé pour détecter un objet (ou alors, dans des cas précis, en récupérant les valeurs d'un pixels aux coordonnées *x,y* afin que la couleur obtenue correspond bien à un objet, et non à la table).

Voici les modifications apportées au programme `balls_tracker.py` pour générer le *threshold* :

```
    ...
    # Threshold the HSV image
    mask = cv2.inRange(hsv, lower_color, upper_color)
    ...


# Test with input from camera
if __name__ == '__main__':
    capture = cv2.VideoCapture('ball3.mp4')
    okay, image = capture.read()
    blur = cv2.GaussianBlur(image, (5,5),0)

    # Convert BGR to HSV
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
    cv2.imshow(WINDOW_NAME, blur)
    h, w, n = np.shape(hsv)

    # Get HSV value from pixel on the center of the frame:
    c = hsv[h//2][w//2]

    # Generate threshold from color:
    lower_color = np.array([c[0] - 10, c[1] - 50, c[2] - 50])
    upper_color = np.array([c[0] + 10, c[1] + 50, c[2] + 50])
    ...
```

Pour la vidéo `ball3.mp4`, il est plus intéressant de récupérer la valeur du pixel à la position $(384, 850)$ pour pouvoir apprendre la couleur de la balle jaune :

```
# Get HSV value from a pixel from the yellow ball:
c = hsv[384][850]
```

Grâce à cette valeur, le programme apprend automatiquement comment localiser la balle jaune dans l'image :

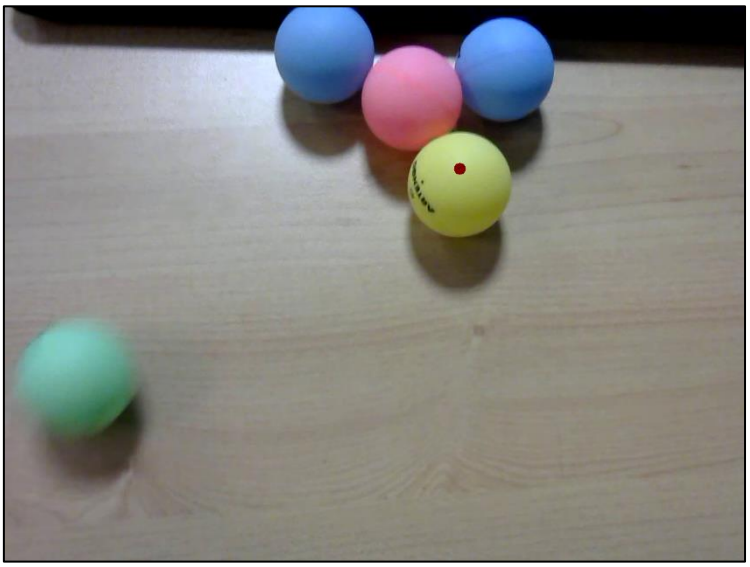


figure 7 – Apprentissage automatique de la couleur de la balle jaune

Conclusion :

Ce TP nous a donc permis d'observer différentes méthodes pour localiser des objets de couleurs dans des images, mais surtout d'en extraire leurs limites.

Tout d'abord, isoler seulement les pixels de la couleur souhaitée se réalise en déterminant les pixels de l'objet qui sont contenus entre des bornes de valeurs. À partir de ces pixels, le centre de l'objet peut être localiser en faisant une moyenne de toutes les positions. Cependant, cette méthode n'est plus efficace lorsque plusieurs objets sont présents.

Ensuite, pour isoler chaque objet d'une image, il est possible de partitionner les groupes de pixels. Néanmoins, il reste à sélectionner le bon objet, ce qui peut se faire en gardant l'objet possédant le plus de pixels détectés, mais il reste la possibilité que l'objet ne reste pas constamment le plus grand.

Enfin, pour éviter que le centre se téléporte entre plusieurs objets, il est nécessaire de vérifier que le centre trouvé est proche de celui de l'image précédente, ce qui permet de s'assurer de la continuité de la position de l'objet.

De plus, les valeurs des bornes n'ont pas besoin d'être définies manuellement : il est possible de réaliser un apprentissage afin de créer les bornes à partir d'un pixel présent sur l'image, ce qui permet de faciliter l'implémentation dans différents environnements.