Rapport Technique

pour

Détection de la couleur des sacs de ciment

Préparé par BENSAID MOHAMED

Ciments du Maroc

January 31, 2024

Contents

I. Introduction

Dans le domaine dynamique de l'efficacité industrielle, ce rapport présente le récit d'un projet crucial mené lors d'un stage en informatique dans une cimenterie. La mission : mettre en œuvre un système de détection des couleurs basé sur Python et OpenCV, déployé sur un Raspberry Pi. Ce système visait à relever un défi pratique : assurer l'identification et l'étiquetage corrects des sacs de ciment au cours du processus de production.

Les erreurs opérationnelles, telles que les erreurs d'étiquetage ou de correspondance entre les produits et les mauvais sacs, peuvent avoir des répercussions importantes dans les environnements industriels. En s'appuyant sur les capacités d'OpenCV, une bibliothèque de vision par ordinateur réputée, et sur l'efficacité compacte d'un Raspberry Pi, notre projet visait à fournir une solution en temps réel à ce défi.

II. la théorie de l'image

II.1. Qu'est-ce qu'une image ?

Une image est une représentation visuelle d'un objet, d'une scène ou d'un concept. Dans le contexte de la technologie numérique, une image est souvent un tableau bidimensionnel de pixels.

II.2. Composition of an Image:

Pixels

Les pixels, abréviation de "picture elements", sont les entités microscopiques qui donnent vie à une image numérique. Il s'agit de pièces de mosaïque qui s'assemblent pour créer un récit visuel cohérent.

Taille et disposition

Disposés méticuleusement en grille, les pixels s'assemblent pour former la tapisserie complexe d'une image. La taille de cette grille, communément appelée résolution, détermine le niveau de détail de l'image.

Résolution

Aspect crucial, la résolution détermine la densité des pixels d'une image. Une résolution plus élevée permet d'obtenir des détails plus fins, essentiels pour préserver la fidélité de la représentation visuelle.

Informations sur les couleurs

Les pixels contribuent non seulement à la complexité spatiale d'une image, mais ils contiennent également des informations essentielles sur les couleurs. Le modèle RGB,

qui comprend les composantes rouge, verte et bleue, détermine la richesse chromatique de chaque pixel.

II.3. Détails sur les pixels

RGB

Le modèle RGB est le langage par lequel les pixels expriment leurs couleurs. En ajustant l'intensité des composantes rouge, verte et bleue, les pixels créent un spectre de teintes. Chaque composante de couleur s'étend généralement de 0 à 255, offrant ainsi une vaste palette pour l'expression artistique.

L'intensité de chaque composante de couleur détermine sa vivacité dans le pixel. Cette interaction dynamique donne lieu à une multiplicité de couleurs et de dégradés qui caractérisent une image visuellement convaincante.

Représentation spatiale

Les coordonnées : Dans le paysage pixellisé, chaque pixel revendique un ensemble unique de coordonnées, définissant son emplacement précis. Cette disposition spatiale structure le récit visuel et en assure la cohérence et la clarté.
Rapport d'aspect : Le rapport d'aspect, attribut fondamental d'une image, influence la forme de la grille de pixels. Ce rapport joue un rôle important dans la composition visuelle et la narration à l'intérieur du cadre.

II.4. Formats de fichiers d'images

La compression

Compression avec perte Certains formats d'image, tels que JPEG, utilisent des techniques de compression avec perte pour équilibrer la taille du fichier et la qualité de l'image. Il en résulte un certain sacrifice de données dans l'intérêt d'un stockage efficace.

Compression sans perte Les formats tels que PNG donnent la priorité à la fidélité, en préservant toutes les données de l'image sans compromis. Si cette approche garantit une qualité irréprochable, elle peut entraîner une augmentation de la taille des fichiers.

Formats les plus courants :

- **JPEG** : réputé pour sa polyvalence, le format JPEG trouve sa place dans le monde de la photographie, où une compression efficace est primordiale.
- **PNG** : adopté pour sa prise en charge de la transparence et son engagement à préserver l'intégrité de l'image grâce à une compression sans perte.
- **GIF** : Format réputé pour son aptitude à transmettre des graphiques simples et des séquences animées avec une palette de couleurs limitée.

II.5. Espaces de couleur : HSV

Définition

En plus du modèle de couleurs RGB largement utilisé, il existe un espace de couleurs alternatif connu sous le nom de HSV, pour Hue, Saturation et Value (teinte, saturation et valeur). Contrairement au RGB, qui est basé sur la combinaison additive des couleurs primaires, le HSV représente les couleurs d'une manière qui s'aligne plus étroitement sur la perception humaine et peut être particulièrement utile dans l'édition et la manipulation d'images.

Composantes du HSV

Hue

- **Définition**: La teinte représente le type de couleur, souvent visualisée sous la forme d'un spectre circulaire où les couleurs se succèdent sans interruption. Elle est mesurée en degrés (de 0 à 360°).
- **Signification**: La teinte nous permet de faire la distinction entre les couleurs telles que le rouge, le vert et le bleu, ce qui permet une compréhension plus intuitive du spectre des couleurs.

Saturation

- **Définition**: La saturation fait référence à l'intensité ou à la vivacité d'une couleur.Elle est mesurée en pourcentage, 0 % correspondant à une échelle de gris (pas de couleur) et 100 % à une saturation totale (couleur pure).
- **Signification**: La saturation influence la richesse et la pureté d'une couleur et permet de décrire des tons subtils ou éclatants

Value

- **Définition**: La valeur représente la luminosité ou l'obscurité d'une couleur. Elle est également mesurée en pourcentage, 0 % correspondant au noir et 100 % au blanc.
- **Signification**: La valeur détermine la clarté ou l'obscurité d'une couleur, ce qui permet de créer des nuances et des teintes.

Comparaison avec RGB

- **Perception humaine**: HSV correspond mieux à la façon dont les humains perçoivent et décrivent les couleurs, ce qui en fait un outil précieux pour les applications centrées sur la couleur.

- **Contrôle artistique**: Alors que le RGB est bien adapté aux écrans numériques, le HSV est privilégié pour le contrôle artistique et les ajustements de couleurs qui imitent l'interprétation humaine.

III. Méthodologie de la détection des couleurs d'une image

III.1. Définition

La détection et la quantification de couleurs spécifiques dans une image est un aspect crucial de l'analyse d'images et de la vision par ordinateur. L'utilisation de l'espace colorimétrique HSV fournit une méthode puissante pour définir et identifier les couleurs cibles. Cette méthode consiste à capturer une image, à la convertir dans l'espace colorimétrique HSV, à définir des plages de couleurs d'intérêt, puis à calculer le nombre de pixels se trouvant dans ces plages de couleurs prédéfinies.

III.2. Étapes de la détection des couleurs d'une image

1. **Acquisition d'images :**Capturez ou importez l'image que vous souhaitez analyser. Assurez-vous que l'image est au format numérique et accessible pour le traitement.

2. **Conversion de l'espace colorimétrique :**Convertissez l'image de son espace colorimétrique natif (généralement RGB) à l'espace colorimétrique HSV. Cette conversion permet une analyse plus intuitive des couleurs, notamment en cas de variations des conditions d'éclairage.

3. **Définition de plages de couleurs :**Établir des plages de couleurs spécifiques dans l'espace HSV qui correspondent aux couleurs d'intérêt. Il s'agit de définir des plages pour la teinte, la saturation et la valeur en fonction des caractéristiques de couleur souhaitées.

4. **Seuil :**Appliquer des techniques de seuillage pour isoler les pixels se trouvant dans les plages de couleurs définies. Ce processus permet d'obtenir une image binaire dans laquelle les pixels situés dans la plage de couleurs spécifiée sont mis en évidence, tandis que les autres sont supprimés. Comptage de pixels :

5. **Comptage de pixels :**Calculer le nombre de pixels qui répondent aux critères de couleur définis. Ce comptage de pixels fournit une mesure quantitative de la prévalence des couleurs ciblées dans l'image.

IV. Implémentation

La mise en œuvre de la détection des couleurs d'image consiste à traduire la méthodologie théorique en étapes pratiques à l'aide d'outils de programmation et de traitement d'images. Vous trouverez ci-dessous un guide étape par étape pour la mise en œuvre de la méthodologie de détection des couleurs d'image présentée précédemment.

IV.1. Hardware et Software

Software

- **Choix du langage de programmation :**

  La lisibilité de Python et ses nombreuses bibliothèques, en particulier OpenCV et NumPy, en font un choix idéal pour la mise en œuvre de la détection des couleurs dans les images.

- **OpenCV pour le traitement d'images :**

  OpenCV fournit une suite complète d'outils pour le traitement d'images, y compris les conversions d'espace colorimétrique et les opérations au niveau du pixel, ce qui simplifie la mise en œuvre des algorithmes de détection des couleurs.

- **NumPy pour les opérations sur les tableaux :**

  Les opérations sur les tableaux de NumPy sont essentielles pour traiter efficacement les données d'image. La nature de NumPy, basée sur les tableaux, s'aligne bien sur la représentation matricielle des pixels d'une image.

Hardware

- **Puissance de calcul intégrée :**

  Le Raspberry Pi 4B, avec ses capacités de calcul améliorées, constitue une plate-forme compacte mais puissante pour exécuter des tâches de traitement d'images.

- **GPIO (General Purpose Input/Output) pour les signaux électriques :**

  Exploitation des pins GPIO du Raspberry Pi pour établir une interface avec des dispositifs externes ou fournir des signaux électriques basés sur les résultats de la détection des couleurs.

- **Extensibilité et module caméra :**

  L'extensibilité du Raspberry Pi 4B, démontrée par ses pins GPIO et sa compatibilité avec les modules de caméra, améliore ses capacités pour les applications de traitement d'images.

IV.2. Importation de bibliothèques

Pour pouvoir utiliser les bibliothèques mentionnées précédemment, il est indispensable de les importer dans notre programme, ce qui permettra au programme de reconnaître et d'utiliser les fonctions des bibliothèques importées.

```
import cv2 #opencv
from picamera2 import Picamera2 #fonctions liees a la camera
import numpy as np
import time #fonctions liees au temps
import RPi.GPIO as GPIO fonctions liees au Signal electrique
```

IV.3. configuration de la caméra

La deuxième étape pour construire ce détecteur de couleur est de faire en sorte que Python lise les images provenant de la caméra afin d'effectuer le processus de détection plus tard, heureusement ceci est pris en charge par la bibliothèque de la caméra importée plus tôt, tout ce que nous avons à faire est de mettre en place quelques configurations.

```
#Appel de la camera
piCam2 = Picamera2()
#definir la resolution et le systeme de couleurs
piCam2.preview_configuration.main.size = (640, 480)
piCam2.preview_configuration.main.format = "RGB888"
#confirmation de la configuration
piCam2.preview_configuration.align()
piCam2.configure("preview")
```

IV.4. Chargement de l'image

Le code ci-dessous permet de démarrer l'appareil photo, de prendre une photo et de l'arrêter.

```
piCam2.start()
frame = piCam2.capture_array()
piCam2.stop()
```

Pour que le programme fonctionne en temps réel, nous devrons mettre en place une boucle qui prendra continuellement des photos avec une certaine condition d'interruption.

Si elle n'est pas arrêtée manuellement, la boucle fonctionnera à l'infini, et aussi vite que le Raspberry Pi peut aller, cela fournira plus d'images que nécessaire, surchargera le système et causera des problèmes de chauffage. Nous pouvons limiter ce problème en définissant une petite fenêtre de temps (réglable) pour mettre la boucle en pause avant de lancer une nouvelle itération.

```
while True:
    frame = piCam2.capture_array()
    #afficher l'image caputre on temps reel
    cv2.imshow("piCam", frame)
    #condition pour arreter la boucle (en cliquons sur q)
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        break
    #Temps de pause
    time.sleep(0.1)
```

IV.5. Prétraitement des images

Nous allons convertir l'espace colorimétrique de l'image de RGB à HSV, ce qui nous permettra d'identifier les couleurs plus facilement.

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

IV.6. Plages de couleurs

Dans cette étape, nous avons défini les plages de couleurs que nous allons identifier, ce qui inclut les 4 couleurs que nous voulons détecter et également la couleur jaune que nous voulons exclure de l'image. Pour ce faire, il suffit de définir des limites inférieures et supérieures, puisque les codes de couleur HSV sont parfaitement ordonnés. Dans opencv la valeur de la teinte est divisée par 2 (de 0-360 à 0-180) et la saturation et la valeur sont multipliées par 2.55 (de 0-100 à 0-255)

```
lower_red = np.array([1, 75, 50])
upper_red = np.array([9, 255, 255])

lower_red2 = np.array([150, 75,50])
upper_red2 = np.array([179, 255, 255])

lower_blue = np.array([90, 75, 75])
upper_blue = np.array([125, 255, 255])

lower_green = np.array([40, 50, 50])
upper_green = np.array([80, 255, 255])


lower_black = np.array([0, 0, 0])
upper_black = np.array([180, 30, 200])

lower_yl = np.array([20, 50, 50])
upper_yl = np.array([40, 255, 255])

mask_red = cv2.inRange(hsv, lower_red, upper_red)
mask_red2= cv2.inRange(hsv, lower_red2, upper_red2)
mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
mask_green = cv2.inRange(hsv, lower_green, upper_green)
mask_black = cv2.inRange(hsv, lower_black, upper_black)
mask_yl = cv2.inRange(hsv, lower_yl, upper_yl)

#elimination de la coleur jaune
inver=cv2.bitwise_not(mask_yl)
fred=cv2.bitwise_and(mask_red,inver)
fred2=cv2.bitwise_and(mask_red2,inver)
fgreen=cv2.bitwise_and(mask_green,inver)
fblue=cv2.bitwise_and(mask_blue,inver)
fblack=cv2.bitwise_and(mask_black,inver)
```

IV.7. Calcule des pixels

Après avoir défini les plages de couleurs, nous calculons le nombre de pixels pour chaque couleur.

```
red_pixels = np.count_nonzero(fred) + np.count_nonzero(fred2)
blue_pixels = np.count_nonzero(fblue)
green_pixels = np.count_nonzero(fgreen)
black_pixels = np.count_nonzero(fblack)
```

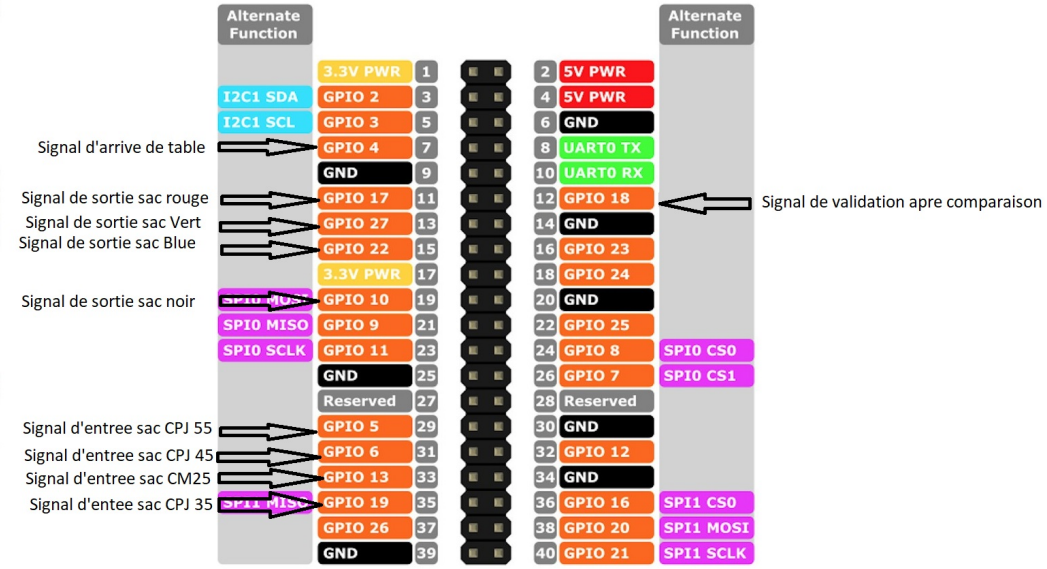

Figure 1.: Schema des pins

IV.8. Définition des signaux d'entrée et de sortie

Pour rendre le détecteur de couleur fonctionnel dans les processus de production, il est très nécessaire de fournir un certain type de communication entre le système que nous avons créé et le système automatisé dans la production, pour rendre cela possible, nous utiliserons des signaux électriques et GPIO.

En Python, nous devons définir les pins qui assureront la communication et spécifier leur fonction.

Le GPIO peut entrer/sortir une tension maximale de 3,3 V, une tension plus élevée peut endommager la carte.

La figure suivante montre toutes les pins utilisées Les pins d'entrée sont réglés pour être nuls à l'aide d'un résistor pull down.

```
#definition des pins de sortie
LED_RED=17
LED_GREEN=27
LED_BLUE=22
LED_BLACK=10
LED_INV=18
#definition des pins d'entree
Button=4
```

```
Red=5 #CPJ55
Blue=6 #CPJ45
Green=13 #CM25
Black=19 #CPJ35
#configuration des pins
GPIO.setmode(GPIO.BCM)

GPIO.setup(LED_RED,GPIO.OUT)
GPIO.setup(LED_GREEN,GPIO.OUT)
GPIO.setup(LED_BLUE,GPIO.OUT)
GPIO.setup(LED_BLACK,GPIO.OUT)
GPIO.setup(LED_INV,GPIO.OUT)

GPIO.setup(Button, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(Red, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(Blue, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(Green, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(Black, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
```

IV.9. Comparaison des pixels

Dans cette partie, le programme compare le nombre de pixels et allume la pin correspondant à la couleur dominante tout en éteignant les autres, si aucune couleur n'est présente, toutes les pins sont éteintes.

```
        if red_pixels > blue_pixels and red_pixels > green_pixels and
                                            red_pixels > black_pixels:
        dominant_color = "Rouge"
        GPIO.output(LED_RED,GPIO.HIGH)
        GPIO.output(LED_GREEN,GPIO.LOW)
        GPIO.output(LED_BLUE,GPIO.LOW)
        GPIO.output(LED_BLACK,GPIO.LOW)

    elif blue_pixels > red_pixels and blue_pixels > green_pixels and
                                          blue_pixels > black_pixels:
        dominant_color = "Blue"
        GPIO.output(LED_RED,GPIO.LOW)
        GPIO.output(LED_GREEN,GPIO.LOW)
        GPIO.output(LED_BLUE,GPIO.HIGH)
        GPIO.output(LED_BLACK,GPIO.LOW)

    elif green_pixels > red_pixels and green_pixels > blue_pixels and
                                          green_pixels > black_pixels:
        dominant_color = "Vert"
        GPIO.output(LED_RED,GPIO.LOW)
        GPIO.output(LED_GREEN,GPIO.HIGH)
        GPIO.output(LED_BLUE,GPIO.LOW)
        GPIO.output(LED_BLACK,GPIO.LOW)


    elif black_pixels > red_pixels and black_pixels > blue_pixels and
                                          black_pixels > green_pixels:
```

```
        dominant_color = "Noir"
        GPIO.output(LED_RED,GPIO.LOW)
        GPIO.output(LED_GREEN,GPIO.LOW)
        GPIO.output(LED_BLUE,GPIO.LOW)
        GPIO.output(LED_BLACK,GPIO.HIGH)
    else:
        dominant_color = "None"
        GPIO.output(LED_RED,GPIO.LOW)
        GPIO.output(LED_GREEN,GPIO.LOW)
        GPIO.output(LED_BLUE,GPIO.LOW)
        GPIO.output(LED_BLACK,GPIO.LOW)
```

IV.10. Lecture du signal d'entrée

Cette fonction lit le signal d'entrée de la qualité et renvoie la couleur correspondante.

```
def insignal():
  if GPIO.input(Red)==GPIO.HIGH:
    color="Rouge"
  elif GPIO.input(Blue)==GPIO.HIGH:
    color="Blue"
  elif GPIO.input(Green)==GPIO.HIGH:
    color="Vert"
  elif GPIO.input(Black)==GPIO.HIGH:
    color="Black"
  else:
    color='None'
  return color
```

IV.11. Comparaison des signaux

Lorsque l'applicateur est prêt à prendre les sacs (détecté par un capteur doté d'une sortie électrique), les signaux d'entrée et de sortie sont comparés, Si la couleur du sac est valide, la sortie de la comparaison est activée.

```
    if GPIO.input(Button)==GPIO.HIGH:
        print(f'{detect_dominant_color(frame2)},{insignal()}')
        if detect_dominant_color(frame2)==insignal() and insignal()!="
                                                    None":
            GPIO.output(LED_INV,GPIO.HIGH)
        else:
            GPIO.output(LED_INV,GPIO.LOW)
```

IV.12. Détection dans un mini cadre

Pour optimiser les systèmes de détection des couleurs, nous pouvons nous concentrer uniquement sur l'endroit où la couleur est présente, en définissant une mini-trame à l'intérieur de la trame originale dont la taille et l'emplacement sont réglables, ce qui réduira également la quantité de données traitées et le nombre d'erreurs, conduisant à un système plus efficace.

```
#la taille du mini-frame
roi_size=100
#la resolution de la camera (a modifier si les paramatres du camera sont
                                    modifie)
height = 480
width = 640
center_x = width // 2
center_y = height // 2
#les 4 points qui dessinent le mini cadre
x1=max(0,center_x-roi_size // 2)
y1=max(0,center_y-roi_size // 2)
x2=min(width,center_x+roi_size // 2)
y2=min(height,center_y+roi_size // 2)
#afficher le mini cadre dans la capture (facultatif)
cv2.rectangle(frame,(x1-1,y1-1),(x2+1,y2+1),(0,0,0),1)
#la nouvelle image generer
frame2=frame[y1:y2,x1:x2]
```

IV.13. Code Complet

```
import cv2
from picamera2 import Picamera2
import numpy as np
import time
import RPi.GPIO as GPIO



LED_RED=17
LED_GREEN=27
LED_BLUE=22
LED_BLACK=10

LED_INV=18

Button=4
Red=5
Blue=6
Green=13
Black=19

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_RED,GPIO.OUT)
GPIO.setup(LED_GREEN,GPIO.OUT)
GPIO.setup(LED_BLUE,GPIO.OUT)
GPIO.setup(LED_BLACK,GPIO.OUT)

GPIO.setup(LED_INV,GPIO.OUT)
GPIO.setup(Button, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(Red, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(Blue, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(Green, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(Black, GPIO.IN,pull_up_down=GPIO.PUD_DOWN)

piCam2 = Picamera2()
piCam2.preview_configuration.main.size = (640, 480)
piCam2.preview_configuration.main.format = "RGB888"
piCam2.preview_configuration.align()
piCam2.configure("preview")
piCam2.start()


def detect_dominant_color(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_red = np.array([1, 75, 50])
    upper_red = np.array([9, 255, 255])

    lower_red2 = np.array([150, 75,50])
    upper_red2 = np.array([179, 255, 255])

    lower_blue = np.array([90, 75, 75])
    upper_blue = np.array([125, 255, 255])
```

```
lower_green = np.array([40, 50, 50])
upper_green = np.array([80, 255, 255])


lower_black = np.array([0, 0, 0])
upper_black = np.array([180, 30, 200])

lower_yl = np.array([20, 50, 50])
upper_yl = np.array([40, 255, 255])


mask_red = cv2.inRange(hsv, lower_red, upper_red)
mask_red2= cv2.inRange(hsv, lower_red2, upper_red2)
mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
mask_green = cv2.inRange(hsv, lower_green, upper_green)
mask_black = cv2.inRange(hsv, lower_black, upper_black)
mask_yl = cv2.inRange(hsv, lower_yl, upper_yl)

# Count the number of non-zero pixels for each color
inver=cv2.bitwise_not(mask_yl)
fred=cv2.bitwise_and(mask_red,inver)
fred2=cv2.bitwise_and(mask_red2,inver)
fgreen=cv2.bitwise_and(mask_green,inver)
fblue=cv2.bitwise_and(mask_blue,inver)
fblack=cv2.bitwise_and(mask_black,inver)

red_pixels = np.count_nonzero(fred) + np.count_nonzero(fred2)
blue_pixels = np.count_nonzero(fblue)
green_pixels = np.count_nonzero(fgreen)
black_pixels = np.count_nonzero(fblack)
print(red_pixels,blue_pixels,green_pixels,black_pixels)
# Determine the dominant color
dominant_color = None
if red_pixels > blue_pixels and red_pixels > green_pixels and
                                    red_pixels > black_pixels:
    dominant_color = "Rouge"
    GPIO.output(LED_RED,GPIO.HIGH)
    GPIO.output(LED_GREEN,GPIO.LOW)
    GPIO.output(LED_BLUE,GPIO.LOW)
    GPIO.output(LED_BLACK,GPIO.LOW)

elif blue_pixels > red_pixels and blue_pixels > green_pixels and
                                    blue_pixels > black_pixels:
    dominant_color = "Blue"
    GPIO.output(LED_RED,GPIO.LOW)
    GPIO.output(LED_GREEN,GPIO.LOW)
    GPIO.output(LED_BLUE,GPIO.HIGH)
    GPIO.output(LED_BLACK,GPIO.LOW)

elif green_pixels > red_pixels and green_pixels > blue_pixels and
                                    green_pixels > black_pixels:
    dominant_color = "Vert"
    GPIO.output(LED_RED,GPIO.LOW)
```

```
        GPIO.output(LED_GREEN,GPIO.HIGH)
        GPIO.output(LED_BLUE,GPIO.LOW)
        GPIO.output(LED_BLACK,GPIO.LOW)


    elif black_pixels > red_pixels and black_pixels > blue_pixels and
                                        black_pixels > green_pixels:
        dominant_color = "Noir"
        GPIO.output(LED_RED,GPIO.LOW)
        GPIO.output(LED_GREEN,GPIO.LOW)
        GPIO.output(LED_BLUE,GPIO.LOW)
        GPIO.output(LED_BLACK,GPIO.HIGH)
    else:
        dominant_color = "None"
        GPIO.output(LED_RED,GPIO.LOW)
        GPIO.output(LED_GREEN,GPIO.LOW)
        GPIO.output(LED_BLUE,GPIO.LOW)
        GPIO.output(LED_BLACK,GPIO.LOW)
    return dominant_color


def insignal():

  if GPIO.input(Red)==GPIO.HIGH:
    color="Rouge"
  elif GPIO.input(Blue)==GPIO.HIGH:
    color="Blue"
  elif GPIO.input(Green)==GPIO.HIGH:
    color="Vert"
  elif GPIO.input(Black)==GPIO.HIGH:
    color="Black"
  else:
    color='None'
  return color


roi_size=100
height = 480
width = 640
center_x = width // 2
center_y = height // 2
x1=max(0,center_x-roi_size // 2)
y1=max(0,center_y-roi_size // 2)
x2=min(width,center_x+roi_size // 2)
y2=min(height,center_y+roi_size // 2)

while True:
    frame = piCam2.capture_array()
    cv2.rectangle(frame,(x1-1,y1-1),(x2+1,y2+1),(0,0,0),1)
    frame2=frame[y1:y2,x1:x2]
    if GPIO.input(Button)==GPIO.HIGH:
        print(f'{detect_dominant_color(frame2)},{insignal()}')
        if detect_dominant_color(frame2)==insignal() and insignal()!="
                                            None":
```

```
            GPIO.output(LED_INV,GPIO.HIGH)
        else:
            GPIO.output(LED_INV,GPIO.LOW)


    cv2.imshow("piCam", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        GPIO.cleanup()
        break
    time.sleep(0.1)

piCam2.stop()
```

IV.14. Execution du programme

Le programme peut fonctionner en deux modes,chaque mode avait son propre fichier de programme et ils sont tous deux stockés à l'emplacement

`/home/cimar/Color_dector`

Le mode manuel

Le mode manuel signifie que nous allons exécuter manuellement les commandes qui lancent le programme. Il dispose d'une fenêtre qui montre le contenu de la caméra à l'utilisateur en temps réel, tout en détectant également les couleurs.Chaque fois que le système redémarre, vous devrez le relancer manuellement.

1. Ouvrir l'interpreteur des commandes

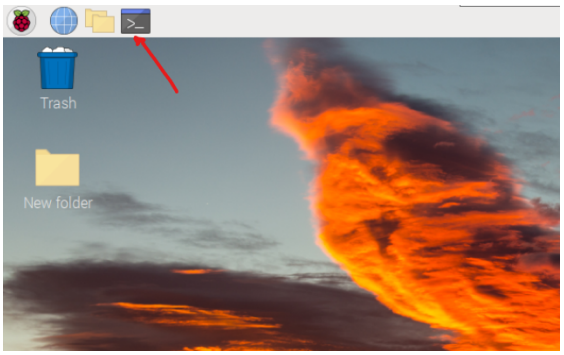

Figure 2.: Enter Caption

2. exécuter cette commande

   `cd Color_dector`

3. exécuter cette commande

```
python avecAffichage.py
```

Le mode de démarrage automatique

Le mode autostart s'exécute automatiquement lorsque le Raspberry Pi démarre. Ce mode s'exécute discrètement en arrière-plan, de sorte qu'il n'affiche pas le contenu de la caméra (mais la détection des couleurs est toujours en cours), ce mode est une version alternative du programme original.

Le mode de démarrage automatique est contrôlé par crontab, qui exécute automatiquement les commandes permettant d'exécuter le programme sans caméra à chaque démarrage du Raspberry Pi.

Pour activer ou desactiver le mode automatique exécuter cette commande

```
sudo crontab -e
```

quand il n'y a pas de # au début de la dernière ligne, cela signifie que l'autostart est activé

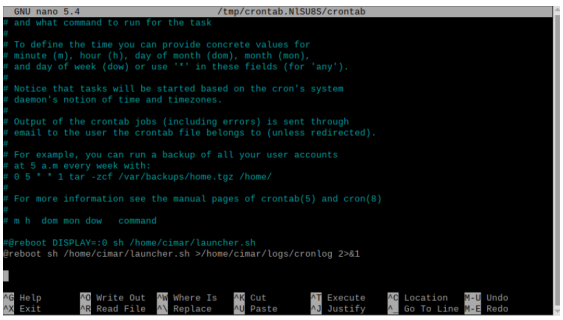

Figure 3.: Démarrage automatique activé

si vous voulez désactiver le # au début de la dernière ligne

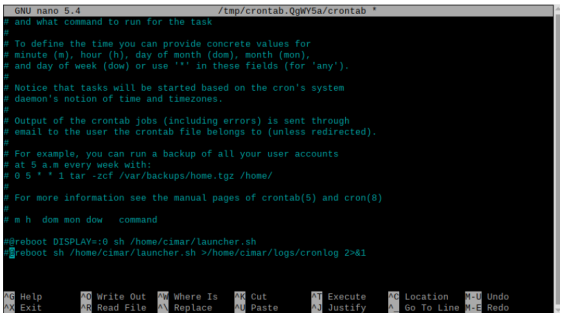

Figure 4.: Démarrage automatique activé

V. Conclusion

En conclusion, l'intégration de Python, OpenCV, NumPy et du Raspberry Pi 4B a permis de créer un puissant système de détection des couleurs des images. Ce système, conçu pour détecter les couleurs des sacs de ciment afin de remplir le produit avec précision, illustre l'efficacité de Python pour le traitement des images et la polyvalence du Raspberry Pi pour les applications du monde réel. La combinaison du logiciel et du matériel crée une solution robuste, démontrant le potentiel du contrôle de qualité automatisé et de la manipulation précise des produits dans les environnements industriels.