	Initiation à l'intelligence artificielle. Application à un trieur de M&Ms	<b>Animation</b>
GIPTIC SII	<i><b>Application au trieur de M&amp;Ms</b></i>	Page 1 sur 44

## Introduction à l'IA avec un trieur de couleurs pour M&M's

### 1. Introduction générale

- **Objectifs du cours :** Comprendre les concepts de base de l'IA et leur application dans un projet concret.
- **Présentation du projet :** Créer un trieur de couleurs pour des M&M's à l'aide d'un Raspberry Pi, avec une progression de techniques (classiques à IA avancée).
- **Rappel des prérequis :**
  - Notions de base en Python.
  - Compréhension de capteurs et d'actionneurs utilisés avec le Raspberry Pi.
  - Contexte des couleurs : spectres lumineux et perception RVB.


### 2. Approche classique : Analyse des écarts de valeurs mesurées

**Principe de la méthode :** Comparer les valeurs RVB mesurées par un capteur à des seuils prédéfinis pour classer les couleurs.

- **Matériel nécessaire :**
  - Raspberry Pi.
  - Capteur de couleur (TCS34725 ou équivalent).
  - Servo-moteur pour trier les M&M's.
- **Étapes :**
  1. **Mesurer les couleurs :** Exploiter le capteur pour obtenir des valeurs RVB des M&M's.
  2. **Définir les seuils :** Établir une gamme de valeurs pour chaque couleur (rouge, vert, bleu, jaune).
  3. **Programmer la logique conditionnelle :** Utiliser des instructions `if-else` pour diriger les M&M's vers des emplacements spécifiques.
- **Limites de l'approche classique :** Sensibilité aux variations de lumière, difficulté à généraliser.

### 3. Introduction aux modèles d'IA classiques

- **Principe :** Remplacer la logique conditionnelle par un modèle d'IA entraîné pour classer les couleurs.
- **Modèles abordés :**
  - Régression logistique.
  - Forêt aléatoire.
  - k-Nearest Neighbors (k-NN).
- **Étapes :**
  1. **Collecte de données :** Créer un dataset en mesurant des valeurs RVB et en annotant manuellement les couleurs.
  2. **Entraînement des modèles :** Utiliser une bibliothèque comme `scikit-learn` pour entraîner un modèle sur les données collectées.
  3. **Évaluation :** Comparer les performances des différents modèles sur des M&M's non vus pendant l'entraînement.
  4. **Implémentation :** Intégrer le modèle choisi dans le programme pour trier les M&M's en temps réel.
- **Discussion :** Avantages de l'IA sur les seuils fixes (meilleure adaptabilité, robustesse).

	Initiation à l'intelligence artificielle. Application à un trieur de M&Ms	<b>Animation</b>
GIPTIC SII	<i><b>Application au trieur de M&amp;Ms</b></i>	Page 2 sur 44

#### 4. Exemple avancé : Introduction au deep learning

- **Principe** : Utiliser un réseau de neurones pour classer les couleurs avec davantage de flexibilité.
- **Matériel et outils** :
  - Raspberry Pi (avec accélérateur comme Google Coral pour le traitement en temps réel, si possible).
  - Bibliothèque TensorFlow ou PyTorch.
- **Étapes** :
  1. **Collecte et augmentation des données** : Capturer un dataset plus large, inclure des variations de luminosité, bruit, etc.
  2. **Création d'un modèle simple** : Un réseau à plusieurs couches (MLP) pour traiter les données RVB.
  3. **Entraînement** : Utiliser un ordinateur pour entraîner le modèle, puis déployer sur le Raspberry Pi.
  4. **Optimisation** : Discuter des outils comme TensorFlow Lite pour optimiser le modèle pour l'embarqué.
  5. **Évaluation finale** : Comparer les performances du deep learning avec les modèles classiques.
- **Discussion** : Applications pratiques du deep learning et limites (temps d'entraînement, besoin en données).

#### 5. Conclusion et ouverture

- **Résumé** :
  - Évolution des techniques : classique → modèles d'IA → deep learning.
  - Avantages et inconvénients de chaque approche.
- **Applications réelles** : Tri automatisé dans l'industrie, reconnaissance d'images, domotique.
- **Prolongements** : Possibilité d'intégrer des caméras pour détecter les formes ou textures en complément des couleurs.
- **Questions et perspectives** : Comment améliorer le modèle avec plus de données ou d'autres capteurs ?

#### 6. Annexes et ressources

- **Code Python pour chaque étape.**
- **Datasets exemples pour tester les modèles.**
- **Références pédagogiques et documentations (scikit-learn, TensorFlow Lite, etc.).**

## 1. INTRODUCTION GENERALE

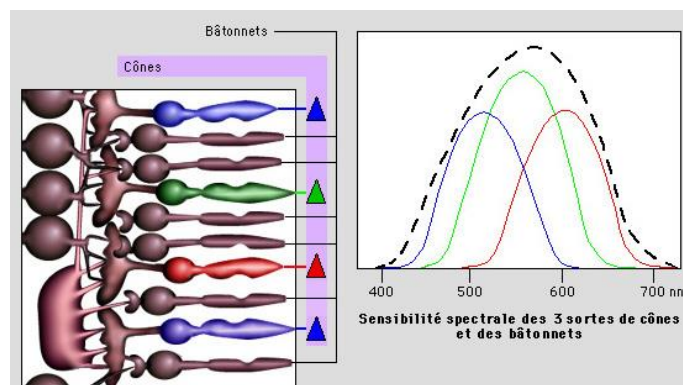
Contexte des couleurs : Spectres lumineux et perception RVB

### 1.1. La nature de la lumière et des couleurs

- **La lumière visible :**  
La lumière est une onde électromagnétique, et la partie visible par l'œil humain correspond à une gamme de longueurs d'onde allant de **400 nm (violet)** à **700 nm (rouge)**.
- **Spectre lumineux :**  
Chaque couleur correspond à une longueur d'onde particulière. Par exemple :
  - Violet : ~400-450 nm
  - Bleu : ~450-495 nm
  - Vert : ~495-570 nm
  - Jaune : ~570-590 nm
  - Rouge : ~620-700 nm
- **Couleurs perçues :**  
La couleur que nous voyons est le résultat de la lumière réfléchiée par un objet. Un objet jaune, par exemple, absorbe les longueurs d'onde dans le bleu et réfléchit celles correspondant au jaune.

### 1.2. La perception des couleurs par l'œil humain

- **Les photorécepteurs :**  
L'œil humain possède trois types de cônes sensibles aux longueurs d'onde :
  - **Cônes L (longueurs d'onde longues, rouge)**
  - **Cônes M (longueurs d'onde moyennes, vert)**
  - **Cônes S (longueurs d'onde courtes, bleu)**



- **Mélange additif des couleurs :**  
L'œil humain perçoit une couleur en combinant les signaux des cônes L, M, et S. Par exemple :
  - Rouge + Vert → Jaune

- Rouge + Bleu → Magenta
- Vert + Bleu → Cyan
- Rouge + Vert + Bleu → Blanc

### 1.3. Modèle RVB (Rouge, Vert, Bleu) RGB

- **Définition :**

Le modèle RGB est un système de représentation des couleurs basé sur la perception humaine. Chaque couleur est définie par une combinaison de trois valeurs (R, G, B), chacune représentant l'intensité lumineuse dans une des trois bandes : rouge, vert, bleu.

- **Plage des valeurs :**

Typiquement, les valeurs RGB vont de 0 (aucune lumière) à 255 (intensité maximale), formant une palette de plus de 16 millions de couleurs.

Exemple :

- Rouge pur : (255, 0, 0)
- Vert pur : (0, 255, 0)
- Bleu pur : (0, 0, 255)
- Blanc : (255, 255, 255)
- Noir : (0, 0, 0)

- **Applications dans le projet :**

Les capteurs de couleur (comme le TCS34725) mesurent les valeurs RVB d'un objet en analysant la lumière réfléchiée par celui-ci et en la décomposant en trois composantes (R, G, B).


### 1.4. Variabilité des mesures RVB

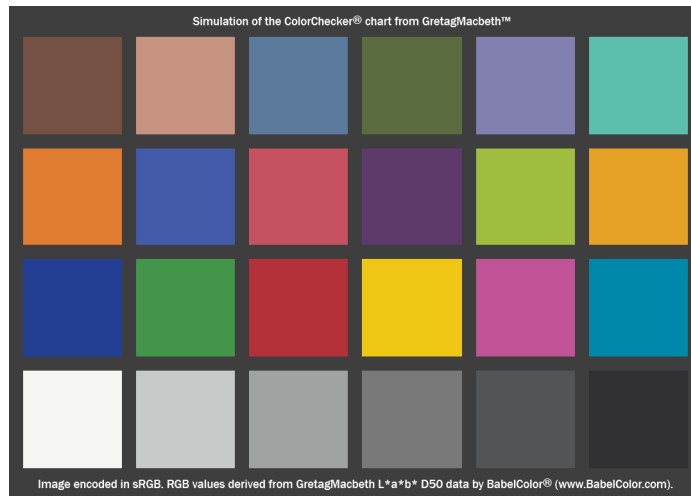
- **Facteurs influençant les mesures :**

- **Source lumineuse :** La couleur perçue peut changer selon la lumière ambiante (naturelle, LED, etc.).
- **Réflectance de la surface :** Les objets brillants ou mats n'interagissent pas de la même manière avec la lumière.
- **Position et distance du capteur :** La mesure peut varier en fonction de l'angle ou de la distance entre le capteur et l'objet.

- **Importance de la calibration :**

Pour garantir des mesures précises et cohérentes, il est souvent nécessaire de calibrer le capteur avec des couleurs de référence avant de commencer le tri.

	Initiation à l'intelligence artificielle. Application à un trieur de M&Ms	<b>Animation</b>
GIPTIC SII	<i><b>Application au trieur de M&amp;Ms</b></i>	Page 5 sur 44



## 1.5. Illustration pratique pour le projet

- **Exemple de mesure :**

Un M&M rouge pourrait donner les valeurs approximatives suivantes :

- Rouge : 200
- Vert : 50
- Bleu : 40

Un M&M jaune, en revanche, pourrait donner :

- Rouge : 180
- Vert : 170
- Bleu : 50

- **Analyse :**

Les valeurs mesurées permettent de distinguer les couleurs des M&M's et de les classer en fonction de seuils ou via des modèles IA.

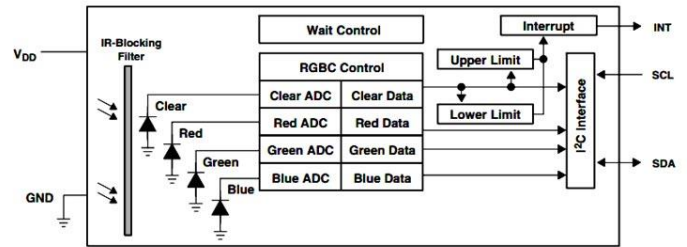
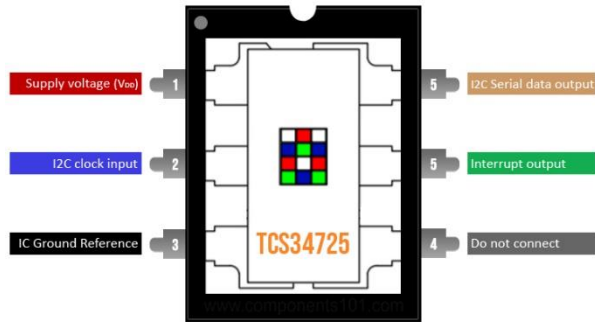
Fonctionnement succinct du capteur de couleur TCS34725

Le TCS34725 est un capteur de couleur populaire pour les projets électroniques, notamment avec un Raspberry Pi ou un Arduino. Voici son fonctionnement résumé :

## 1.6. Principe de base

- Détection de lumière réfléchi :**

Le capteur utilise un photodétecteur pour mesurer la lumière réfléchi par un objet. Il capte les intensités lumineuses dans trois bandes spectrales principales : **rouge (R)**, **vert (G)**, et **bleu (B)**, ainsi qu'une composante **luminance totale** (canal clair).



- Éclairage intégré :**

Le TCS34725 possède une LED blanche intégrée pour fournir un éclairage constant, améliorant la précision des mesures indépendamment des conditions lumineuses ambiantes.

## 1.7. Composants principaux

- Filtre passe-bande :**

Un filtre intégré isole les longueurs d'onde correspondant aux couleurs rouge, verte, et bleue.

- Photodiodes :**

Chaque photodiode est dédiée à une bande de fréquence (R, G, B, et clair). Elles transforment la lumière incidente en signaux électriques proportionnels à l'intensité lumineuse.

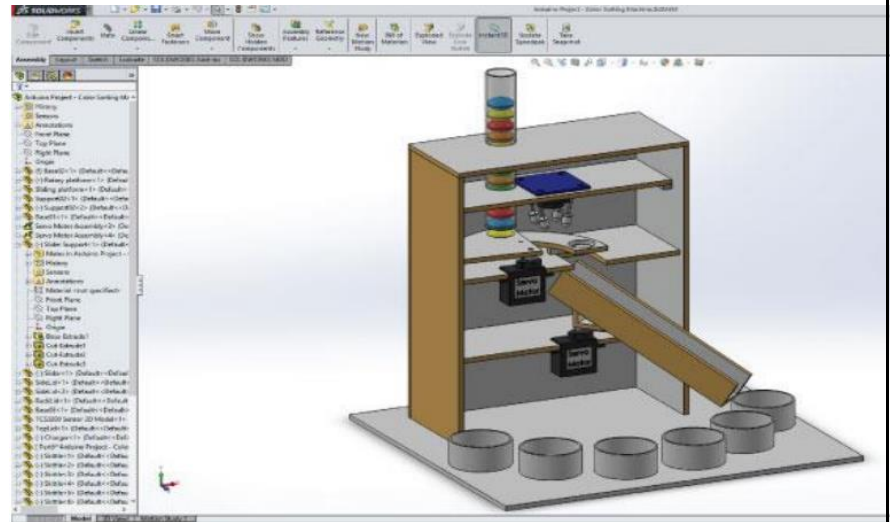
- Convertisseur analogique-numérique (ADC) :**

Le capteur convertit les signaux analogiques en valeurs numériques exploitables via un microcontrôleur ou un ordinateur (Raspberry Pi, par exemple).



## 1.8. Interface et communication

- **Protocole I2C :**  
Le TCS34725 communique via le protocole I2C, ce qui facilite son intégration avec des microcontrôleurs ou des ordinateurs monocarte comme le Raspberry Pi.
- **Commande logicielle :**  
Les bibliothèques comme Adafruit TCS34725 (en Python) simplifient son utilisation en fournissant des fonctions pour lire les valeurs RVB et calibrer le capteur.



## 1.9. Avantages

Pour les grand fan de M&M's

- Compact, précis et facile à utiliser.
- LED intégrée pour garantir des mesures constantes.
- Compatible avec de nombreux environnements de développement (Arduino, Raspberry Pi).

## 1.10 Installation des paquets

```
pip install smbus2
sudo apt install -y python3-pyqt5 python3-opengl
python -m pip install scipy
python -m pip install numpy
python -m pip install matplotlib
python -m pip install sklearn
python -m pip install pandas
pip3 install -U scikit-learn scipy matplotlib
pip install pandas tensorflow scikit-learn joblib
```

## 2. APPROCHE CLASSIQUE :


### ANALYSE DES ECARTS DE VALEURS MESUREES

On utilise pour répondre au cahier des charge deux servo moteur :

Les servos sont câblés sur les sortie GPIO 17 bas et 18 haut

1. **Servo 1** : avec trois positions :
  - Position 1 : Chargement.
  - Position 2 : Mesure.
  - Position 3 : Éjection.



	Initiation à l'intelligence artificielle. Application à un trieur de M&Ms	Animation
GIPTIC SII	<b>Application au trieur de M&amp;Ms</b>	Page 8 sur 44

## 2. Servo 2 :

- 6 positions pour trier les M&M's par couleur : marron, vert, orange, jaune, rouge, bleu.

Pour une meilleure gestion des servos moteurs utilisation de la librairie pigpiod pour les raspberry < P5

<https://abyz.me.uk/rpi/pigpio/download.html>

```
wget https://github.com/joan2937/pigpio/archive/master.zip
unzip master.zip
cd pigpio-master
make
sudo make install
```

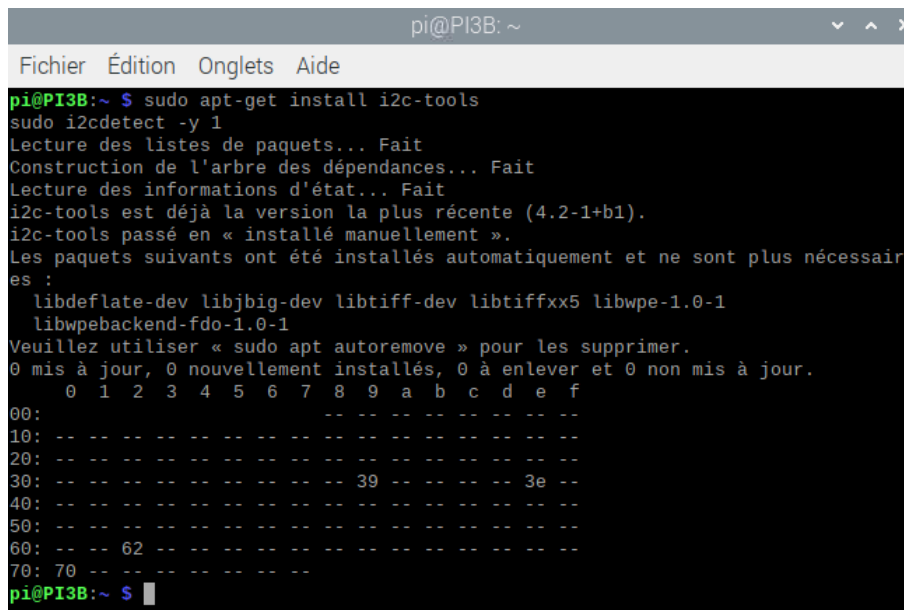
## Bus I2C

Le capteur de couleur sur le bus I2C, activer le bus I2C pour le raspberry et vérifier la présence du capteur

`sudo apt-get install i2c-tools`

`sudo i2cdetect -y 1`

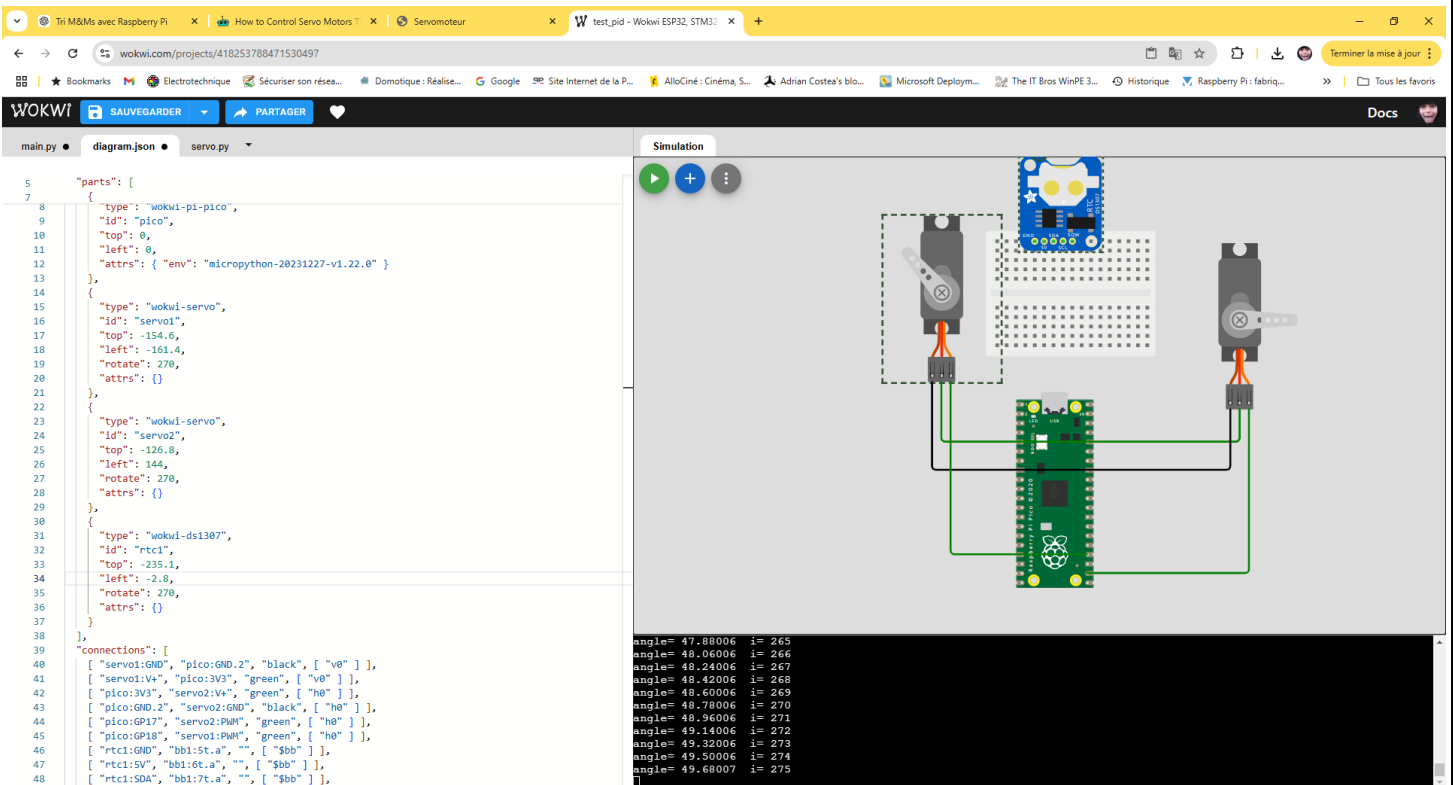
adresse #39 pour APDS-9960 Digital Proximity, Ambient Light, RGB and Gesture Sensor



```
pi@PI3B: ~
Fichier  Édition  Onglets  Aide
pi@PI3B:~$ sudo apt-get install i2c-tools
sudo i2cdetect -y 1
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
i2c-tools est déjà la version la plus récente (4.2-1+b1).
i2c-tools passé en « installé manuellement ».
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
  libdeflate-dev libjbig-dev libtiff-dev libtiffxx5 libwpe-1.0-1
  libwpebackend-fdo-1.0-1
Veuillez utiliser « sudo apt autoremove » pour les supprimer.
0 mis à jour, 0 nouvellement installés, 0 à enlever et 0 non mis à jour.
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  39  --  --  --  --  3e  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  62  --  --  --  --  --  --  --  --  --  --  --  --  --
70: 70  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
pi@PI3B:~$
```

## 2.1 Câblage et simulation exemple en Micropython et WOKWI





The screenshot shows the Wokwi web IDE interface. On the left, a Python script is visible, defining parts like 'wokwi-pi-pico', 'wokwi-servo', and 'wokwi-ds1307', and their connections. On the right, a simulation window shows a Raspberry Pi Pico connected to two servo motors. Below the simulation, a log window displays the servo motor's position and pulse width over time.

## 2.2 Création d'un fichier CSV pour les échantillons

Programme pour collecter des échantillons de données et générer un fichier CSV contenant les valeurs RGB mesurées pour 10 pastilles de chaque couleur. Le programme suit un cycle itératif où vous insérez les pastilles par couleur et validez chaque échantillon. Le fichier CSV sera généré après la collecte de tous les échantillons sous le nom :

couleurs\_echantillons.csv

### Programme Python

```
from gpiozero import Servo
from time import sleep
from gpiozero.pins.pigpio import PiGPIOFactory
import smbus
import csv

factory = PiGPIOFactory()

servoh = Servo(18, min_pulse_width=0.56, max_pulse_width=1.85,
pin_factory=factory)
servob = Servo(17, min_pulse_width=0.52, max_pulse_width=1.8 ,
pin_factory=factory)

# Configuration I2C
bus = smbus.SMBus(1)
```

```
SENSOR_ADDRESS = 0x39
```

```
# Définir le chemin du fichier CSV
nom_fichier = "couleurs.csv"
```

```
# Liste des couleurs et structure pour stocker les données
COULEURS = ["bleu", "rouge", "vert", "marron", "jaune", "orange"]
echantillons = {couleur: [] for couleur in COULEURS}
```

```
# Initialisation du capteur
```

```
def init_sensor():
    bus.write_byte_data(SENSOR_ADDRESS, 0x00 | 0x80, 0x03)
    bus.write_byte_data(SENSOR_ADDRESS, 0x07 | 0x80, 0x00)
```

```
# Lecture des données RGB
```

```
def read_colors():
    bus.write_byte_data(SENSOR_ADDRESS, 0x00 | 0x80, 0x03)
    sleep(0.5)
    data = bus.read_i2c_block_data(SENSOR_ADDRESS, 0x10 | 0x80, 8)
    green = data[1] * 256 + data[0]
    red = data[3] * 256 + data[2]
    blue = data[5] * 256 + data[4]
```

```
# ajustement pour fond noir
red -= 19.00
green -= 35
blue -= 16
```

```
return red, green, blue
```

```
# Collecte des données pour une couleur spécifique
```

```
def collect_sample(couleur, num_samples=10):
    print(f"Insérez des pastilles de couleur '{couleur}'.")
    for i in range(num_samples):
        print(f"Mesure {i + 1}/{num_samples} pour {couleur}.")
        print("Position de mesure...")
        servoh.mid()
        sleep(1)
        red, green, blue = read_colors()
        echantillons[couleur].append((red, green, blue))
        print(f"R: {red}, G: {green}, B: {blue} enregistré pour {couleur}.")

    print("Position de dechargement...")
    servoh.max()
    sleep(1)

    print("Position de chargement...")
    servoh.min()
```

sleep(3)

```
# Enregistrement des données dans un fichier CSV
def save_to_csv():
    with open(nom_fichier, 'w', newline='', encoding='utf-8') as
fichier_csv:
        file = csv.writer(fichier_csv)
        file.writerow(["Couleur", "Red", "Green", "Blue"]) # En-têtes
        for couleur, samples in echantillons.items():
            for red, green, blue in samples:
                file.writerow([couleur, red, green, blue])
        print(f"Données enregistrées dans {nom_fichier}.")

# Programme principal
def main():
    try:
        init_sensor()
        servoh.mid()
        servob.mid()
        for couleur in COULEURS:
            collect_sample(couleur)
            save_to_csv()
    except KeyboardInterrupt:
        print("Programme interrompu.")
    finally:
        servoh.value = None
        servob.value = None
        print("Servos désactivés. Fin du programme.")

if __name__ == "__main__":
    main()
```

## Fonctionnement

1. **Initialisation :**
  - Le capteur et les servos sont initialisés dans les positions par défaut.
2. **Collecte des Échantillons :**
  - Pour chaque couleur dans la liste COULEURS, le programme lit les données RGB pour 10 pastilles (ou une valeur ajustable dans num\_samples).
  - Entre chaque lecture, le servo horizontal (servoh) est déplacé pour simuler la position de mesure de déchargement et de chargement.
3. **Enregistrement des Données :**
  - Une fois toutes les pastilles mesurées, les données sont enregistrées dans un fichier CSV avec des colonnes pour la couleur et les valeurs RGB.
4. **Arrêt Sécurisé :**
  - Les servos sont désactivés et retournent à leur état neutre à la fin ou en cas d'interruption.

## Détermination des plages de couleurs à partir des échantillons

## Programme Python

```
import csv

# Fonction pour calculer les plages de variation des couleurs
def calculate_color_ranges(file_path):
    color_data = {}

    # Lecture du fichier CSV
    with open(file_path, 'r', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile)
        next(reader) # Ignorer les en-têtes
        for row in reader:
            r, g, b, color = float(row[0]), float(row[1]),
float(row[2]), row[3]
            if color not in color_data:
                color_data[color] = {"R": [], "G": [], "B": []}
            color_data[color]["R"].append(r)
            color_data[color]["G"].append(g)
            color_data[color]["B"].append(b)

    # Calcul des plages pour chaque couleur
    color_ranges = {}
    for color, channels in color_data.items():
        color_ranges[color] = {
            "R": (min(channels["R"]), max(channels["R"])),
            "G": (min(channels["G"]), max(channels["G"])),
            "B": (min(channels["B"]), max(channels["B"]))
        }

    return color_ranges

# Fonction principale
def main():
    file_path = "couleurs_echantillons.csv" # Remplacez par le
chemin de votre fichier CSV
    color_ranges = calculate_color_ranges(file_path)

    print("Plages de variation des couleurs :")
    for color, ranges in color_ranges.items():
        print(f"{color.capitalize()} :")
        print(f"  R (Rouge) : {ranges['R']}")
        print(f"  G (Vert) : {ranges['G']}")
        print(f"  B (Bleu) : {ranges['B']}")

# Exécution du script
if __name__ == "__main__":
    main()
```

## Fonctionnement

- **Lecture des données CSV :**

Le fichier CSV est lu ligne par ligne, et les valeurs RGB associées à chaque couleur sont stockées dans un dictionnaire.

- **Calcul des plages de variation :**

Pour chaque couleur, les valeurs minimales et maximales de R, G et B sont déterminées.

- **Affichage des résultats :**

Les plages de variation pour chaque couleur sont imprimées de manière lisible.

## Résultat

	R		G		B	
	min	max	min	max	min	max
Color						
bleu	0.0	1.0	0.0	1.0	4.0	5.0
jaune	10.0	12.0	10.0	12.0	2.0	2.0
marron	1.0	1.0	-2.0	-1.0	0.0	0.0
orange	8.0	10.0	3.0	4.0	1.0	1.0
rouge	6.0	7.0	0.0	1.0	0.0	1.0
vert	1.0	2.0	4.0	5.0	2.0	2.0

## Utilisation

```
COLOR_RANGES = {
    "bleu": [(0.0, 0.0), (1.0, 2.0), (4.0, 5.0)],
    "rouge": [(6.0, 7.0), (0.0, 1.0), (1.0, 1.0)],
    "vert": [(2.0, 2.0), (5.0, 6.0), (2.0, 2.0)],
    "marron": [(0.0, 1.0), (-1.0, -1.0), (0.0, 1.0)],
    "jaune": [(10.0, 12.0), (10.0, 13.0), (2.0, 3.0)],
    "orange": [(9.0, 11.0), (4.0, 5.0), (1.0, 2.0)],
}
```

## Réalisation du programme de tri en prenant en compte des variations avec une pondération de 1

### Programme Python

```
'''
lancer

sudo pigpiod

avant
'''

from gpiozero import Servo
```

```

from time import sleep

from gpiozero.pins.pigpio import PiGPIOFactory

import smbus
import time
import csv
import joblib

factory = PiGPIOFactory()

servoh = Servo(18, min_pulse_width=0.56/1000,
max_pulse_width=1.85/1000, pin_factory=factory)
servob = Servo(17, min_pulse_width=0.52/1000,
max_pulse_width=1.8/1000, pin_factory=factory)

servob_POSITIONS = {
    "marron": -0.3,      # Position pour marron
    "vert": 0.6,         # Position pour vert
    "orange": -0.95,     # Position pour orange
    "jaune": -0.7,       # Position pour jaune
    "rouge": 0.1,        # Position pour rouge
    "bleu": 0.9          # Position pour bleu
}

COLOR_RANGES = {
    "bleu": [(0.0, 0.0), (1.0, 2.0), (4.0, 5.0)],
    "rouge": [(6.0, 7.0), (0.0, 1.0), (1.0, 1.0)],
    "vert": [(2.0, 2.0), (5.0, 6.0), (2.0, 2.0)],
    "marron": [(0.0, 1.0), (-1.0, -1.0), (0.0, 1.0)],
    "jaune": [(10.0, 12.0), (10.0, 13.0), (2.0, 3.0)],
    "orange": [(9.0, 11.0), (4.0, 5.0), (1.0, 2.0)],
}

# Get I2C bus
bus = smbus.SMBus(1)

# Initialisation du capteur
def init_sensor():
    # Initialisation du capteur
    # TCS3414 address, 0x39(57)
    # Select control register, 0x00(00), with Command register,
    0x80(128)
    #          0x03(03)  Power ON, ADC enable
    bus.write_byte_data(0x39, 0x00 | 0x80, 0x03)

```

```
# Lecture des données de couleur
def read_colors():
    global rang
    bus.write_byte_data(0x39, 0x00 | 0x80, 0x03)
    # TCS3414 address, 0x39(57)
    # Select gain register, 0x07(07), with Command register,
    0x80(128)
    #          0x00(00) Gain : 1x, Prescaler Mode = Divide by 1
    bus.write_byte_data(0x39, 0x07 | 0x80, 0x00)
    time.sleep(0.5)
    # TCS3414 address, 0x39(57)
    # Read data back from 0x10(16), 8 bytes, with Command register,
    0x80(128)
    # Green LSB, Green MSB, Red LSB, Red MSB
    # Blue LSB, Blue MSB, cData LSB, cData MSB
    data = bus.read_i2c_block_data(0x39, 0x10 | 0x80, 8)
    # Convert the data
    green = data[1] * 256 + data[0]
    red = data[3] * 256 + data[2]
    blue = data[5] * 256 + data[4]
    cData = data[7] * 256 + data[6]
    # Calculate luminance
    luminance = (-0.32466 * red) + (1.57837 * green) + (-0.73191 *
blue)

    red -= 19.00
    green -= 35.00
    blue -= 16.00

    return cData, red, green, blue

# Déterminer la couleur dominante
def detect_color():
    _, r, g, b = read_colors()
    print(f"R: {r}, G: {g}, B: {b}")
    tolerance = 1
    for color, (r_range, g_range, b_range) in COLOR_RANGES.items():
        if (
            (r_range[0] - tolerance) <= r <= (r_range[1] +
tolerance) and
            (g_range[0] - tolerance) <= g <= (g_range[1] +
tolerance) and
            (b_range[0] - tolerance) <= b <= (b_range[1] +
tolerance)
        ):
            return color
    return "inconnu"
```



```
# Trier les M&M's
def tri_m_and_ms():
    while True:
        print("Passage en position de chargement...")
        servoh.min()
        sleep(1)
        print("Passage en position de mesure...")
        servoh.mid()
        couleur = detect_color()
        print(f"Couleur détectée : {couleur}")
        # Pause pour éviter des lectures rapides
        sleep(1)
        if couleur in servob_POSITIONS:
            print(f"Tri pour {couleur}...")
            servob.value = servob_POSITIONS[couleur]
            #control_servo(SERVO1_GPIO,
SERVO1_POSITIONS["ejection"]) # Éjection après tri
            sleep(0.5)
            servoh.max()
        else:
            print("Couleur inconnue, ignorer l'objet")
        # Pause pour éviter des lectures rapides
        sleep(1)
        servob.value = 0

# Programme principal
print("Start in the middle")
servoh.mid()
servob.mid()
rang=1
try:
    init_sensor()
    tri_m_and_ms()
except KeyboardInterrupt:
    print("Arrêt du programme.")
```

## Fonctionnement

1. **Servo 1 :**
  - Change entre 3 positions (chargement, mesure, éjection) pour gérer le flux des M&M's.
  - Les angles pour chaque position sont définis dans max mid et min.
2. **Servo 2 :**
  - Positionné selon la couleur détectée : marron, vert, orange, jaune, rouge, ou bleu.
  - Les angles pour chaque couleur sont définis dans SERVO2\_POSITIONS.
3. **Détection des couleurs :**

- Les couleurs sont détectées en comparant les valeurs de rouge, vert et bleu. Des seuils sont définis pour certaines couleurs complexes comme jaune, orange ou marron.

#### 4. Tri :

- Le premier servo gère le déplacement des M&M's entre les étapes.
- Le second servo place les M&M's dans le compartiment correspondant.

### Ajustements et Tests

- **Calibrage des seuils de couleur** : Ajustez les seuils selon les valeurs spécifiques des couleurs obtenues par votre capteur.
- **Angles des servos** : Modifiez les valeurs pour les adapter à votre configuration matérielle.

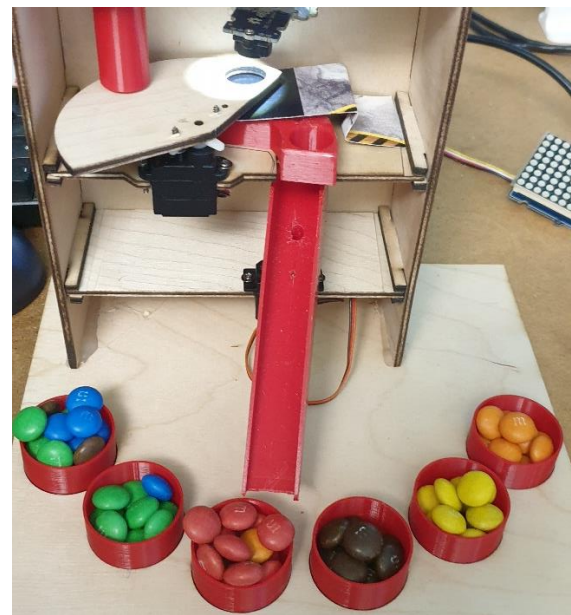
### Constations avec une tolérance de :

**0 les couleurs ont du mal à être détectées en fonction des variations d'éclairage**

**1 le tri est parfait aucune erreur, fin de la manipe (non !)**



**Avec une tolérance de 2 il y a une confusion entre les bleus et vert et entre les rouges et oranges.**



### 3. INTRODUCTION AUX MODELES D'IA CLASSIQUES

#### • 3.1 Principe :

Remplacer la logique conditionnelle par un modèle d'IA entraîné pour classer les couleurs.

#### • 3.2 Modèles abordés :

- Régression logistique. LR
- Forêt aléatoire.
- k-Nearest Neighbors (k-NN).

##### 3.2.1. Régression logistique

###### Principe

- La régression logistique est un modèle statistique utilisé pour résoudre des problèmes de classification binaire ou multi-classes.
- **Fonctionnement** : Elle modélise la probabilité qu'une observation appartienne à une classe donnée en utilisant une fonction logistique (sigmoïde).
- **Sortie** : Les probabilités que l'entrée appartienne à chaque classe. La classe avec la plus grande probabilité est choisie.

###### Avantages

- Simplicité, rapidité, facile à implémenter.
- Bonne performance sur des données linéairement séparables.

### Inconvénients

- Ne gère pas bien les relations non linéaires entre les caractéristiques.
- Sensible aux caractéristiques fortement corrélées.

### 3.2.2. Forêt aléatoire (Random Forest)

#### Principe

- La forêt aléatoire est un ensemble de **plusieurs arbres de décision**, chacun entraîné sur un sous-ensemble aléatoire des données (méthode de bootstrap) et un sous-ensemble aléatoire des caractéristiques.
- **Fonctionnement** : Chaque arbre vote pour une classe, et la classe majoritaire est choisie comme prédiction.

#### Étapes principales

1. **Création des arbres** :
  - Les arbres sont créés en sélectionnant un sous-ensemble aléatoire des échantillons et des caractéristiques.
  - Chaque arbre est construit de manière indépendante.
2. **Prédiction** : Lorsqu'un M&M est mesuré, chaque arbre effectue une prédiction. La classe finale est celle qui obtient le plus de votes.
3. **Importance des caractéristiques** : La forêt aléatoire peut également identifier quelles caractéristiques (R, G, B) sont les plus importantes pour la classification.

#### Avantages

- Robuste aux sur-apprentissages grâce à l'agrégation.
- Peut modéliser des relations complexes entre les caractéristiques.
- Pas besoin de normaliser les données d'entrée.

#### Inconvénients

- Plus lent que des modèles simples comme la régression logistique.
- Moins interprétable qu'un seul arbre de décision.

### 3.2.3. k-Nearest Neighbors (k-NN)

#### Principe

- Le k-NN est un modèle basé sur les instances. Il classe un nouvel échantillon en fonction des **k échantillons les plus proches** dans les données d'entraînement.
- **Fonctionnement** : Il calcule les distances entre l'échantillon à classer et toutes les observations de l'ensemble d'entraînement, puis vote pour la classe la plus fréquente parmi les k plus proches voisins.

#### Étapes principales

1. **Choix de kkk** : kkk est un hyperparamètre qui détermine le nombre de voisins à considérer. Des valeurs courantes sont 3, 5 ou 7.
2. **Distance** : La distance est souvent calculée à l'aide de la distance euclidienne :  

$$d = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}$$
3. **Classification** : Les kkk voisins les plus proches sont identifiés, et l'étiquette majoritaire est attribuée.

#### Avantages

- Facile à comprendre et à implémenter.
- Aucune phase d'entraînement explicite, ce qui le rend très rapide en préparation.

#### Inconvénients

- Coût computationnel élevé pour les prédictions sur de grands ensembles de données (doit calculer la distance avec toutes les instances).
- Sensible à l'échelle des caractéristiques (les données doivent être normalisées si les échelles diffèrent).

## Conclusion

- **Régression logistique** : Utilisée en première approche si les données sont simples et bien séparées.
- **Forêt aléatoire** : Idéal pour des problèmes plus complexes avec des interactions entre les caractéristiques.
- **k-NN** : Bon pour débiter ou si le jeu de données est petit, mais moins adapté pour des ensembles volumineux.

Ces trois modèles permettent une introduction progressive à l'IA et montrent comment des approches différentes peuvent résoudre un même problème de classification.

On peut tester également

LDA

Naïve Bayes

SVM

### 3.2.4 Analyse discriminante linéaire (LDA)

- **Classe utilisée** : `LinearDiscriminantAnalysis`
- **Principe** : LDA trouve des combinaisons linéaires de caractéristiques qui maximisent la séparation entre les classes.
- **Avantage** : Convient aux problèmes où les distributions des classes suivent une loi normale.

- **Utilisation** : Bonne alternative à la régression logistique lorsque les classes sont bien séparées.

### 3.2.5 Machine à vecteurs de support (SVM)

- **Classe utilisée** : SVC
- **Gamma** : Détermine l'influence d'un seul point d'entraînement. Avec `gamma='auto'`, cette influence est ajustée automatiquement.
- **Principe** : Cherche à maximiser la marge entre les classes en utilisant des hyperplans. Peut être étendu à des problèmes non linéaires avec des noyaux (kernel).
- **Avantage** : Performant pour les données complexes.
- **Inconvénient** : Lent pour les grands jeux de données.

### 3.2.6 Naïve Bayes (NB)

- **Classe utilisée** : GaussianNB
- **Principe** : Utilise le théorème de Bayes en supposant que les caractéristiques sont indépendantes (hypothèse naïve).
- **Particularité** : Convient aux données où chaque classe suit une distribution gaussienne.
- **Avantage** : Rapide et performant pour des données simples.
- **Inconvénient** : Hypothèse d'indépendance souvent irréaliste.

### Résumé : Quand utiliser quel modèle ?

Modèle	Complexité	Données linéaires	Données non linéaires	Interprétabilité	Rapidité
Régression logistique	Faible	Oui	Non	Élevée	Rapide
LDA	Moyenne	Oui	Non	Moyenne	Moyenne
k-NN	Faible	Oui	Oui	Moyenne	Lent
Arbre de décision	Moyenne	Oui	Oui	Élevée	Moyenne
Naïve Bayes	Faible	Oui	Non	Moyenne	Rapide
SVM	Élevée	Oui	Oui (avec noyaux)	Faible	Lent

Ces modèles permettent une large exploration des approches supervisées pour différents types de données et contraintes.

### 3.3. Quatre étapes à réaliser :

1. **Collecte de données** : Créer un dataset en mesurant des valeurs RVB et en annotant manuellement les couleurs.
2. **Entraînement des modèles** : Utiliser une bibliothèque comme `scikit-learn` pour entraîner un modèle sur les données collectées.
3. **Évaluation** : Comparer les performances des différents modèles sur des M&M's non vus pendant l'entraînement.

4. **Implémentation** : Intégrer le modèle choisi dans le programme pour trier les M&M's en temps réel.

### **Etape N°1 collecte des données**

Voir 2.2 Création d'un fichier CSV pour les échantillons

### **Etape N°2 Entraînement des modèles**

On choisit six modèles LDA, LR, KNN, CART, NB, et SVM

### **Réalisation du programme en python Apprentissage Ai**

```
import pandas as pd
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.preprocessing import LabelEncoder
import joblib

# === 1. Chargement et exploration des données ===

# Chargement du dataset
file_path = 'couleurs.csv' # Remplacez par le chemin correct
try:
    dataset = pd.read_csv(file_path)
    print("Dataset loaded successfully.")
except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found.")
    exit()

# Dimensions du dataset
print("Dataset dimensions:", dataset.shape)

# Aperçu des premières lignes
print("First 5 rows:\n", dataset.head())

# Statistiques descriptives
print("Descriptive statistics:\n", dataset.describe())

# Matrice de dispersion des variables
```



```

scatter_matrix(dataset)
plt.title("Scatter Matrix of Dataset")
plt.show()

# === 2. Préparation des données ===

# Séparation des caractéristiques (RVB) et des labels (classe des
couleurs)
X = dataset.iloc[:, :3].values # Colonnes RVB
Y = dataset.iloc[:, 3].values # Classe de couleur

# Encodage des labels (classes des couleurs)
label_encoder = LabelEncoder()
Y_encoded = label_encoder.fit_transform(Y)

# Séparation des données en ensembles d'entraînement et de validation
X_train, X_validation, Y_train, Y_validation = train_test_split(
    X, Y_encoded, test_size=0.30, random_state=1
)

# === 3. Définition et évaluation des modèles ===

# Liste des modèles à tester
models = [
    ('LR', LogisticRegression(solver='liblinear', multi_class='ovr')),
    ('LDA', LinearDiscriminantAnalysis()),
    ('KNN', KNeighborsClassifier()),
    ('CART', DecisionTreeClassifier()),
    ('NB', GaussianNB()),
    ('SVM', SVC(gamma='auto'))
]

# Comparaison des modèles
results = []
names = []
print("\nModel Evaluation Results:")
for name, model in models:
    kfold = StratifiedKFold(n_splits=7, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print(f"{name}: {cv_results.mean():.6f} ({cv_results.std():.6f})")

# Visualisation des performances
plt.boxplot(results, labels=names)
plt.title('Algorithm Comparison')
plt.ylabel('Accuracy')
plt.show()

```

```
# === 4. Entraînement, prédictions et sauvegarde des modèles ===

# Fonction utilitaire pour entraîner, prédire et sauvegarder un modèle
def train_and_save_model(model, model_name, X_train, Y_train,
X_validation, Y_validation, label_encoder):
    model.fit(X_train, Y_train)
    predictions = model.predict(X_validation)

    # Sauvegarde du modèle et de l'encodeur
    joblib.dump(model, f'{model_name}.pkl')
    joblib.dump(label_encoder, f'label_encoder_{model_name}.pkl')

    # Évaluation des performances
    print(f"\n=== Results for {model_name} ===")
    print("Accuracy:", accuracy_score(Y_validation, predictions))
    print("Confusion Matrix:\n", confusion_matrix(Y_validation,
predictions))
    print("Classification Report:\n", classification_report(Y_validation,
predictions))

# Entraînement et sauvegarde pour chaque modèle
for name, model in models:
    train_and_save_model(model, name, X_train, Y_train, X_validation,
Y_validation, label_encoder)

# === Fin du programme ===
print("All models trained and saved successfully.")
```

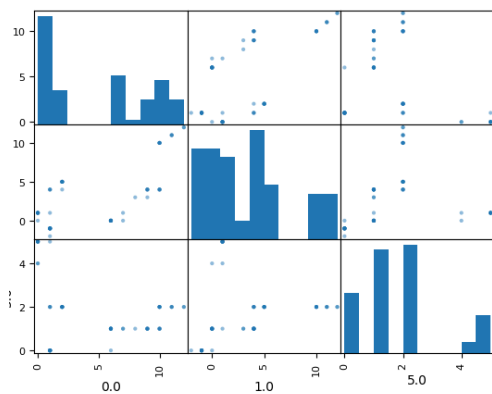
1. **Chargement des données** : Ajout de gestion d'erreur pour s'assurer que le fichier CSV existe avant de continuer.
2. **Exploration des données** : Ajout d'une matrice de dispersion pour visualiser les relations entre les caractéristiques.
3. **Préparation des données** : Utilisation explicite de `iloc` pour séparer les caractéristiques et les labels.
4. **Comparaison des modèles** : Résultats affichés avec moyenne et écart type. Les graphiques de comparaison sont ajoutés.
5. **Entraînement et sauvegarde** : Fonction utilitaire pour éviter la duplication de code, avec des évaluations détaillées pour chaque modèle.

Les résultats sont les suivants

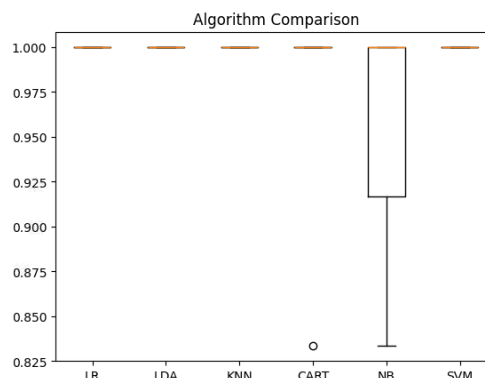
LR : 1.000000 (0.000000)  
LDA: 1.000000 (0.000000)  
KNN: 1.000000 (0.000000)  
CART: 0.976190 (0.058321)  
NB: 0.952381 (0.075292)

SVM: 1.000000 (0.000000)

Matrice de dispersion des variables



Visualisation des performances



Les résultats affichés montrent une excellente performance des modèles testés sur le jeu de données.

### 3.4. Performances des modèles :

- **Régression Logistique (LR) :**
  - Score moyen d'exactitude (accuracy) : **1.000000** avec écart-type **0.000000**.
  - Cela signifie que le modèle a classé toutes les instances correctement, sans variation dans les résultats.
- **Analyse Discriminante Linéaire (LDA) :**
  - Score d'exactitude : **1.000000 (0.000000)**.
  - Comme pour LR, ce modèle est parfait sur ce jeu de données.
- **k-Nearest Neighbors (KNN) :**
  - Score d'exactitude : **1.000000 (0.000000)**.
  - Ce modèle, basé sur les voisins les plus proches, a également obtenu une performance parfaite.
- **Arbre de Décision (CART) :**
  - Score moyen : **0.976190** avec un écart-type de **0.058321**.
  - Ce modèle a montré une très légère baisse de performance et une variabilité dans les prédictions, probablement en raison de la structure de l'arbre ou de petites erreurs de classification.
- **Naïve Bayes (NB) :**
  - Score moyen : **0.952381** avec un écart-type de **0.075292**.
  - Ce modèle montre une précision un peu plus faible, probablement due à l'hypothèse d'indépendance entre les caractéristiques qui n'est pas totalement respectée.
- **Support Vector Machines (svm) :**
  - Score moyen : **1.000000 (0.000000)**.
  - Performances parfaites, confirmant la capacité des SVM à bien séparer les données.

### 3.5. Analyse des métriques globales :

- **Accuracy globale : 1.0**
  - Cela signifie que toutes les instances du jeu de test ont été correctement classées par le modèle final utilisé.

Matrice de confusion :

```
[2 0 0 0 0 0]
[0 4 0 0 0 0]
[0 0 4 0 0 0]
[0 0 0 3 0 0]
[0 0 0 0 1 0]
[0 0 0 0 0 4]
```

- Chaque case de la diagonale (par ex., 2 pour la classe 0, 4 pour la classe 1, etc.) montre que toutes les instances de chaque classe ont été correctement prédites.
- L'absence de valeurs hors de la diagonale indique qu'il n'y a eu aucune erreur de classification.

#### 4. Rapport de classification :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2
1	1.00	1.00	1.00	4
2	1.00	1.00	1.00	4
3	1.00	1.00	1.00	3
4	1.00	1.00	1.00	1
5	1.00	1.00	1.00	4

- **Precision, Recall et F1-score :**
  - Tous les scores sont à **1.00**, ce qui reflète une classification parfaite.
- **Support :** Le nombre d'échantillons par classe (ex. 2 pour la classe 0, 4 pour la classe 1, etc.).

## Conclusions :

1. **Performance globale :**
  - Les modèles LR, LDA, KNN et SVM ont montré une précision parfaite (accuracy = 1.0).
  - CART et NB ont légèrement moins bien performé, mais restent très proches d'une classification parfaite.
2. **Robustesse :**
  - L'absence d'erreurs dans la matrice de confusion et la cohérence des scores (précision, rappel, F1-score) montrent que les modèles gèrent bien ce jeu de données.
3. **Limites potentielles :**
  - Les performances parfaites pourraient indiquer un surapprentissage si le jeu de données est trop simple ou pas assez diversifié.
  - Il serait important de tester ces modèles sur un jeu de données plus varié ou réaliste pour évaluer leur robustesse dans des situations réelles.
4. **Choix du modèle :**
  - Si les ressources sont limitées, LR ou KNN sont simples et performants.
  - Pour des applications nécessitant une plus grande adaptabilité, SVM ou CART pourraient être envisagés.

## 3.6 Réalisation du programme en python de tri en fonction des prédictions Tri Ai

```
from gpiozero import Servo
from time import sleep
from gpiozero.pins.pigpio import PiGPIOFactory
import smbus
import time
import csv
import joblib
import pandas as pd

# Configuration des servos avec la bibliothèque GPIOZero et
PiGPIOFactory
factory = PiGPIOFactory()

servoh = Servo(18, min_pulse_width=0.56/1000,
max_pulse_width=1.85/1000, pin_factory=factory)
servob = Servo(17, min_pulse_width=0.52/1000,
max_pulse_width=1.8/1000, pin_factory=factory)

# Charger le modèle et le label encoder
modele = joblib.load('LR.pkl')
label_encoder = joblib.load('label_encoder_LR.pkl')

# Définir les positions des servos
servoh_POSITIONS = {
    "chargement": 0.56/1000, # Position de chargement
    "mesure": 0.645/1000,    # Position de mesure
    "ejection": 1.85/1000    # Position d'éjection
}

servob_POSITIONS = {
    "jaune": -0.95, # Position pour jaune
    "marron": -0.3, # Position pour marron
    "rouge": 0.1,   # Position pour rouge
    "orange": -0.7, # Position pour orange
    "vert": 0.6,    # Position pour vert
    "bleu": 0.98    # Position pour bleu
}

# Configuration du bus I2C
bus = smbus.SMBus(1)

# Initialisation du capteur de couleur
def init_sensor():
    """Initialisation du capteur TCS3414."""
    bus.write_byte_data(0x39, 0x00 | 0x80, 0x03)

# Lecture des données de couleur
def read_colors():
```

```

"""Lit les valeurs RGB et de luminosité à partir du capteur
TCS3414."""
bus.write_byte_data(0x39, 0x00 | 0x80, 0x03)
bus.write_byte_data(0x39, 0x07 | 0x80, 0x00)
time.sleep(0.5)

data = bus.read_i2c_block_data(0x39, 0x10 | 0x80, 8)
green = data[1] * 256 + data[0]
red = data[3] * 256 + data[2]
blue = data[5] * 256 + data[4]
cData = data[7] * 256 + data[6]

red -= 19
green -= 33
blue -= 16

return cData, red, green, blue

# Déterminer la couleur dominante
def detect_color():
    """Utilise le modèle pour prédire la couleur à partir des valeurs
    RGB."""
    _, red, green, blue = read_colors()
    print(f"R: {red}, G: {green}, B: {blue}")

    nouvelle_couleur = pd.DataFrame([[red, green, blue]],
    columns=["Red", "Green", "Blue"])
    couleur_encoded = modele.predict(nouvelle_couleur)
    couleur = label_encoder.inverse_transform([couleur_encoded[0]])[0]

    print(f"Couleur prédite : {couleur}")
    return couleur

# Trier les M&M's
def tri_m_and_ms():
    """Trie les M&M's selon leur couleur prédite."""
    while True:
        print("Passage en position de chargement...")
        servoh.min()
        sleep(1)

        print("Passage en position de mesure...")
        servoh.mid()
        couleur = detect_color()
        print(couleur)

        if couleur in servob_POSITIONS:
            print(f"Tri pour {couleur}...")
            servob.value = servob_POSITIONS[couleur]

```

```

        sleep(0.5)
        servoh.max()
    else:
        print("Couleur inconnue, ignorer l'objet")

    sleep(1)
    servob.value = 0

# Programme principal
print("Initialisation...")
servoh.mid()
servob.mid()

try:
    init_sensor()
    tri_m_and_ms()
except KeyboardInterrupt:
    print("Arrêt du programme.")

```

## Objectif

Le programme contrôle un système de tri de M&M's basé sur la détection de couleurs à l'aide d'un Raspberry Pi, un capteur de couleur, et des servomoteurs pour orienter les M&M's vers différentes positions en fonction de leur couleur prédite.

## Structure

1. **Initialisation des périphériques :**
  - Utilise `gpiozero` pour contrôler les servomoteurs (un pour la position verticale, un pour la rotation horizontale).
  - Initialise le bus I2C pour communiquer avec le capteur de couleur TCS3414.
2. **Chargement des modèles d'apprentissage automatique :**
  - Charge un modèle de classification (LogisticRegression) et son encodeur de labels à partir de fichiers `.pkl`.
3. **Définition des positions des servomoteurs :**
  - Mappe des couleurs prédéfinies à des positions spécifiques des servomoteurs pour orienter les M&M's.
4. **Fonctionnement du capteur :**
  - **`init_sensor`** : Initialise le capteur de couleur.
  - **`read_colors`** : Lit les données brutes du capteur (RVB, luminance) et applique des corrections.
5. **Prédiction des couleurs :**
  - **`detect_color`** : Prédiction basée sur les valeurs RVB mesurées par le capteur, avec un modèle ML et un label encoder pour décoder les prédictions en noms de couleurs.
6. **Triage des M&M's :**
  - **`tri_m_and_ms`** : Cycle continu :
    - Positionne le M&M pour mesure.
    - Prédit sa couleur.



- Oriente le servomoteur correspondant à la couleur prédite.
- Repositionne le système pour le prochain M&M.

#### 7. Gestion des interruptions :

- Permet l'arrêt propre du programme avec Ctrl+C.

### Points clés

- **Modularité** : Chaque étape (lecture capteur, prédiction, tri) est isolée dans une fonction.
- **Flexibilité** : Les positions des servomoteurs et les couleurs sont configurables.
- **Apprentissage automatique** : Utilise un modèle préentraîné pour des prédictions plus robustes que des seuils fixes.

### 3.7 TEST DES MODELS

#### Régression logistique. LR

```
# Charger le modèle et le label encoder
Exploitationmodele = joblib.load('LR.pkl')
label_encoder =
joblib.load('label_encoder_LR.pkl')
```



Confusion rouge et orange un marron dans les verts

#### Analyse Discriminante Linéaire (LDA)

```
# Charger le modèle et le label encoder
modele = joblib.load('DA.pkl')
label_encoder =
joblib.load('label_encoder_LDA.pkl')
```




Tri OK pas d'erreur

#### k-Nearest Neighbors (KNN) :

```
modele = joblib.load('KNN.pkl')
label_encoder =
joblib.load('label_encoder_KNN.pkl')
```

#### Naïve Bayes (NB) :

```
modele = joblib.load('NB.pkl')
label_encoder =
joblib.load('label_encoder_NB.pkl')
```

	Initiation à l'intelligence artificielle. Application à un trieur de M&Ms	<b>Animation</b>
GIPTIC SII	<b>Application au trieur de M&amp;Ms</b>	Page 31 sur 44



**Tri OK pas d'erreur**



Confusion vert, marron et bleu un vert dans les oranges

Arbre de Décision (**CART**) :

```

modele = joblib.load('KNN.pkl')
label_encoder =
joblib.load('label_encoder_KNN.pkl')

```



Confusion marron et rouge, un vert dans les oranges

Support Vector Machines (**SVM**)

```

modele = joblib.load('SVM.pkl')
label_encoder =

```



joblib.load('label\_encoder\_SVM.pkl')

**Tri OK pas d'erreur**

EN pratique les modèles Analyse Discriminante Linéaire (LDA), k-Nearest Neighbors (KNN) et Support Vector Machines (SVM) : donnent 100% de résultats

Seul le model LR ne correspond pas aux prédictions.

Nous pouvons aussi vérifier la performance avec un dataset différent ou ajouter des variations (lumière, bruit) pour confirmer ces résultats.

## 4.INTRODUCTION AUX RESEAUX DE NEURONES

**Principe :** Utiliser un réseau de neurones pour classer les couleurs avec davantage de flexibilité.

Etapas principale :

1. Préparation des données
2. Création du model avec Tensorflow/Keras
3. Entraînement du modèle
4. Evaluation du modèle

## 5. Utilisation du model pour prédire des couleurs

### 4.1 Préparation des données

```
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Charger les données à partir du fichier CSV
fichier_csv = "couleurs.csv" # Remplacez par le chemin de votre fichier CSV
data = pd.read_csv(fichier_csv)

# Préparer les données
X = data[["Red", "Green", "Blue"]].values # Les colonnes RGB
y = data["Couleur"].values # La colonne des étiquettes

# Encoder les étiquettes en entiers
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Convertir les étiquettes en format one-hot (nécessaire pour TensorFlow)
y_one_hot = to_categorical(y_encoded)

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.2, random_state=42)
```

### 4.2 Création du model

```
# Définir le modèle TensorFlow

model = Sequential([
    Dense(64, input_dim=3, activation='relu'), # Couche d'entrée
    Dropout(0.2), # Dropout pour éviter le surapprentissage
    Dense(32, activation='relu'), # Couche cachée
    Dense(y_one_hot.shape[1], activation='softmax') # Couche de sortie
])

# Compiler le modèle
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

### 4.3 Entraîner le model

```
# Entraîner le modèle
print("Entraînement du modèle...")
```

```
history = model.fit(X_train, y_train, epochs=50, batch_size=8, validation_data=(X_test, y_test))
```

```
# Évaluer le modèle
```

```
print("\nÉvaluation sur l'ensemble de test :")
```

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```
print(f"Précision : {test_accuracy * 100:.2f}%")
```

```
# Charger l'historique d'entraînement pour visualisation
```

```
import matplotlib.pyplot as plt
```

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
```

```
plt.legend()
```

```
plt.show()
```

```
# Sauvegarder le modèle entraîné
```

```
model.save("modele_tensorflow.h5")
```

```
print("Modèle sauvegardé sous 'modele_tensorflow.h5'.")
```

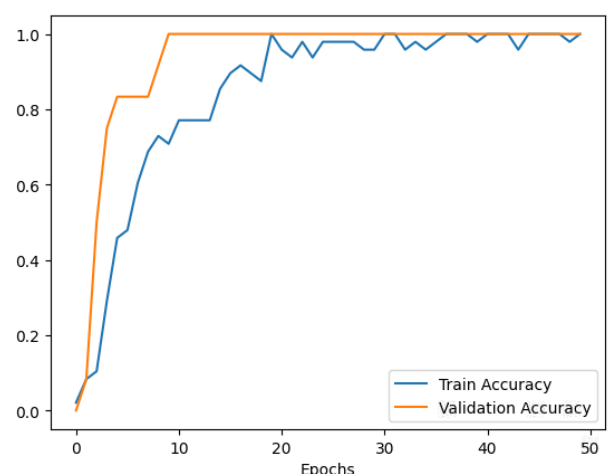
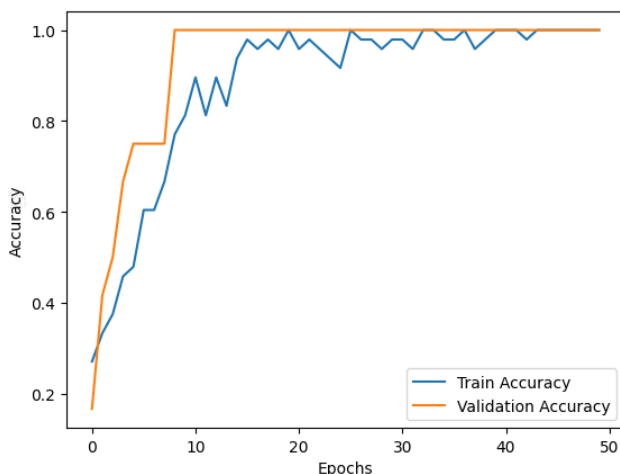
```
# Sauvegarder l'encodeur des étiquettes
```

```
import joblib
```

```
joblib.dump(label_encoder, "label_encoder_tensorflow.pkl")
```

```
print("Encodeur des étiquettes sauvegardé sous 'label_encoder_tensorflow.pkl'.")
```

. Deux entraînements avec le même fichier pour 10 mesures par couleur



L'interprétation des courbes **accuracy** et **val\_accuracy** dans un graphique représentant l'entraînement et la validation d'un modèle machine learning est essentielle pour évaluer la performance et détecter des problèmes comme le sur-apprentissage ou le sous-apprentissage.

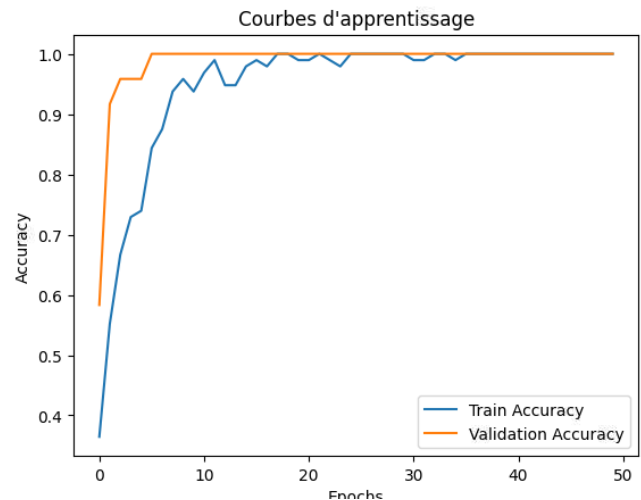
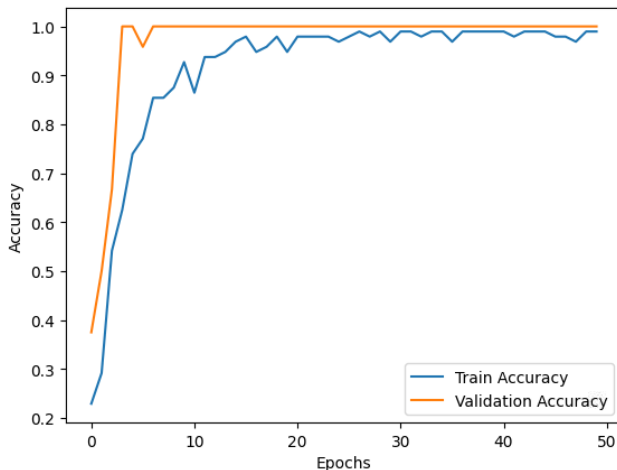
Définitions

1. **Accuracy** : Précision sur les données d'entraînement. Elle indique la proportion des prédictions correctes effectuées par le modèle sur les données d'entraînement.
2. **Val\_accuracy** : Précision sur les données de validation. Elle mesure la capacité du modèle à généraliser sur des données qu'il n'a pas vues pendant l'entraînement.

- Avec 20 valeurs par couleurs en séquences couleurs avec API

Avec 20 valeurs par

4.4



## Programme de tri avec tensor flow

'''

Pré-requis :

- Lancer la commande `sudo pigpiod` avant d'exécuter ce script.
- Assurez-vous que les bibliothèques nécessaires sont installées :
  - tensorflow
  - gpiozero
  - smbus
  - pandas

'''

```
from gpiozero import Servo
from time import sleep
from gpiozero.pins.pigpio import PiGPIOFactory
import smbus
import time
import pandas as pd
import tensorflow as tf
import joblib
```

```
# Configuration du Raspberry Pi
factory = PiGPIOFactory()
```

```
# Initialisation des servomoteurs
servo_horizontal = Servo(18, min_pulse_width=0.56/1000, max_pulse_width=1.85/1000,
pin_factory=factory)
servo_vertical = Servo(17, min_pulse_width=0.52/1000, max_pulse_width=1.7/1000,
pin_factory=factory)
```

```
# Charger le modèle TensorFlow
```

```

modele = tf.keras.models.load_model('modele_tensorflow.h5')
# Charger l'encodeur de labels
label_encoder = joblib.load('label_encoder_tensorflow.pkl')

# Définir les positions des servomoteurs pour chaque couleur
SERVO_POSITIONS = {
    "horizontal": {
        "chargement": -0.98,
        "mesure": 0,
        "ejection": 0.98
    },
    "vertical": {
        "jaune": -0.94,
        "orange": -0.7,
        "marron": -0.3,
        "rouge": 0.1,
        "vert": 0.55,
        "bleu": 0.95
    }
}

# Initialisation du bus I2C pour le capteur de couleur
bus = smbus.SMBus(1)

# Fonction : Initialiser le capteur de couleur
def init_sensor():
    bus.write_byte_data(0x39, 0x00 | 0x80, 0x03) # Activer le capteur
    print("Capteur initialisé.")

# Fonction : Lire les données de couleur depuis le capteur
def read_colors():
    bus.write_byte_data(0x39, 0x07 | 0x80, 0x00) # Configuration du gain
    time.sleep(0.5)
    data = bus.read_i2c_block_data(0x39, 0x10 | 0x80, 8) # Lecture des données
    red = data[3] * 256 + data[2]
    green = data[1] * 256 + data[0]
    blue = data[5] * 256 + data[4]

    # Calibrage des couleurs pour corriger les déviations
    red = max(0, red - 19)
    green = max(0, green - 33)
    blue = max(0, blue - 16)

    print(f"Valeurs RGB : R={red}, G={green}, B={blue}")
    return red, green, blue

# Fonction : Détecter la couleur dominante
def detect_color():
    red, green, blue = read_colors()

```



```
# Préparer les données pour le modèle TensorFlow
nouvelle_couleur = [[red, green, blue]] # Forme attendue : liste de listes
prediction = modele.predict(nouvelle_couleur)
couleur_index = tf.argmax(prediction[0]).numpy() # Index de la classe prédite
couleur = label_encoder.inverse_transform([couleur_index])[0] # Décoder l'index
print(f"Couleur détectée : {couleur}")
return couleur
```

# Fonction : Positionner les servomoteurs

```
def position_servo(servo, position):
```

```
    servo.value = position
```

```
    sleep(0.5)
```

# Fonction : Trier les M&M's

```
def tri_m_and_ms():
```

```
    while True:
```

```
        print("Déplacement en position de chargement...")
```

```
        position_servo(servo_horizontal, SERVO_POSITIONS["horizontal"]["chargement"])
```

```
        print("Déplacement en position de mesure...")
```

```
        position_servo(servo_horizontal, SERVO_POSITIONS["horizontal"]["mesure"])
```

```
        couleur = detect_color()
```

```
        if couleur in SERVO_POSITIONS["vertical"]:
```

```
            print(f"Tri pour la couleur : {couleur}")
```

```
            position_servo(servo_vertical, SERVO_POSITIONS["vertical"][couleur])
```

```
            print("Éjection...")
```

```
            position_servo(servo_horizontal, SERVO_POSITIONS["horizontal"]["ejection"])
```

```
        else:
```

```
            print("Couleur inconnue, objet ignoré.")
```

```
        sleep(1) # Pause avant la prochaine itération
```

# Programme principal

```
if __name__ == "__main__":
```

```
    try:
```

```
        print("Initialisation du capteur...")
```

```
        init_sensor()
```

```
        print("Début du tri...")
```

```
        tri_m_and_ms()
```

```
    except KeyboardInterrupt:
```

```
        print("Programme arrêté par l'utilisateur.")
```

```
    finally:
```

```
        # Désactiver les servomoteurs
```

```
        servo_horizontal.value = None
```

```
        servo_vertical.value = None
```

```
        print("Servos désactivés.")
```



## 4.5 Résultats obtenu avec TensorFlow sur une base de 120 échantillons

Il reste une confusion entre quelques orange et jaune mais le tri est presque parfait

Model Séquentiel



Model API

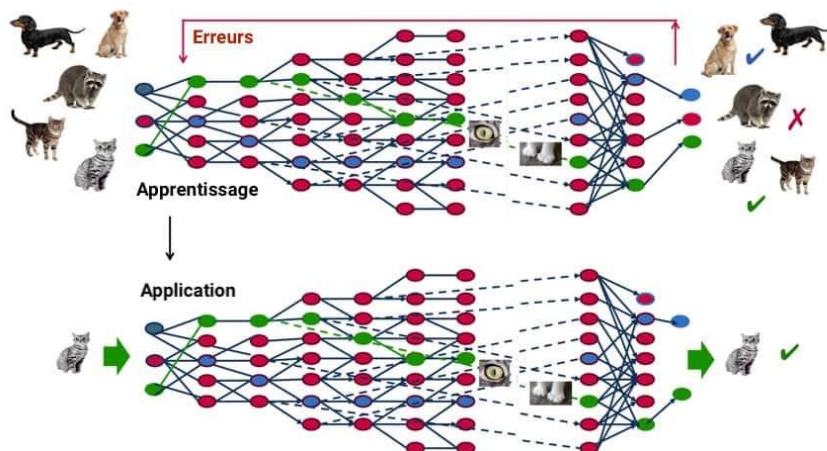


## 5. Entrainement en DeepLearning

Le deep learning est une sous-catégorie de l'intelligence artificielle qui repose sur des réseaux de neurones. Ces réseaux sont particulièrement adaptés pour apprendre des représentations complexes à partir des données brutes, sans nécessiter de règles explicites ou d'étiquetage manuel.

Dans le cadre d'un trieur de couleurs :

- Plutôt que de définir manuellement les seuils RVB ou d'étiqueter des échantillons pour chaque couleur, un réseau de neurones peut apprendre à distinguer les couleurs directement à partir de l'analyse des données brutes capturées par un capteur ou une caméra.
- Ce type de modèle excelle dans des situations où les couleurs se chevauchent ou où les variations de lumière rendent difficile l'utilisation d'approches traditionnelles.



### 5.1. Chargement et Préparation des Données

- Les données sont chargées depuis un fichier CSV contenant des valeurs RGB. Ces données représentent des couleurs (chaque ligne correspond à une couleur avec des valeurs Red, Green, et Blue).
- Les classes (étiquettes) sont générées automatiquement en utilisant les combinaisons uniques des valeurs RGB. Cela crée une correspondance entre les combinaisons RGB et les indices de classe.

- Les étiquettes sont ensuite converties en format **one-hot encoding** pour être utilisées dans un modèle de classification.

#### 5.2. Division des Données

- Les données sont divisées en deux ensembles : un ensemble d'entraînement (80%) et un ensemble de test (20%), pour entraîner et évaluer le modèle.

#### 5.3. Construction du Modèle

- Un modèle **Sequential** est défini avec les couches suivantes :
  - Dense(64, relu)** : Une couche dense (fully connected) avec 64 neurones et activation ReLU.
  - Dropout(0.2)** : Une régularisation Dropout pour réduire le surapprentissage.
  - Dense(32, relu)** : Une deuxième couche dense avec 32 neurones.
  - Dense(output, softmax)** : Une couche de sortie avec un nombre de neurones correspondant au nombre de classes (softmax pour une classification multiclasse).

#### 5.4. Compilation et Entraînement

- Le modèle est compilé avec :
  - L'optimiseur **Adam**.
  - Une fonction de perte **categorical\_crossentropy** (adaptée au problème de classification multiclasse).
  - Une métrique d'évaluation : **accuracy**.
- Le modèle est entraîné sur 50 époques avec une taille de batch de 8.

#### 5.5. Évaluation et Visualisation

- Le modèle est évalué sur l'ensemble de test pour mesurer sa précision.
- Les performances d'entraînement et de validation (accuracy) sont tracées pour analyser la convergence du modèle.

#### 5.6. Sauvegarde des Résultats

- Le modèle entraîné est sauvegardé sous le format .h5.
- L'encodeur des étiquettes (association entre les classes et les combinaisons RGB) est également sauvegardé pour une utilisation future.

#### 5.7 Objectif du Programme

Le but est d'entraîner un modèle de classification capable d'identifier des couleurs (ou classes) à partir de leurs combinaisons RGB. Ce programme pourrait être utilisé dans des applications comme la reconnaissance de couleurs ou des systèmes basés sur des palettes colorimétriques.

```
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import joblib
import numpy as np

# Charger les données depuis le fichier CSV
fichier_csv = "couleurs_deep.csv" # Remplacez par le chemin de votre fichier CSV
data = pd.read_csv(fichier_csv)

# Les données ne contiennent pas de colonne de labels explicites
# On suppose que chaque combinaison RGB correspond à une classe unique
# Charger les données sans la colonne des labels explicites
X = data.values # Les colonnes RGB
```

```
# S'assurer que les données sont de type float
X = X.astype(float)

# Vérifier et gérer les valeurs manquantes
if pd.isnull(X).any():
    X = pd.DataFrame(X).fillna(0).values

# Générer des étiquettes basées sur des combinaisons uniques de RGB
y = pd.factorize([tuple(rgb) for rgb in X])[0]

# Convertir les étiquettes en format one-hot
y_one_hot = to_categorical(y)

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.2, random_state=42)

# Définir le modèle TensorFlow
model = Sequential([
    Dense(64, input_dim=3, activation='relu'), # Couche d'entrée avec 3 neurones pour RGB
    Dropout(0.2), # Dropout pour éviter le surapprentissage
    Dense(32, activation='relu'), # Couche cachée
    Dense(y_one_hot.shape[1], activation='softmax') # Couche de sortie
])

# Compiler le modèle
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Entraîner le modèle
print("Entraînement du modèle...")
history = model.fit(X_train, y_train, epochs=50, batch_size=8, validation_data=(X_test, y_test))
# Convertir les étiquettes one-hot en indices
y_test_indices = np.argmax(y_test, axis=1) # Convertir les étiquettes one-hot en indices

# Afficher les indices des classes
print(f"Indices des classes dans les données d'entraînement : {sorted(set(y_test_indices))}")

# Évaluer le modèle
print("\nÉvaluation sur l'ensemble de test :")
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Précision : {test_accuracy * 100:.2f}%")

# Visualisation de l'historique d'entraînement
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

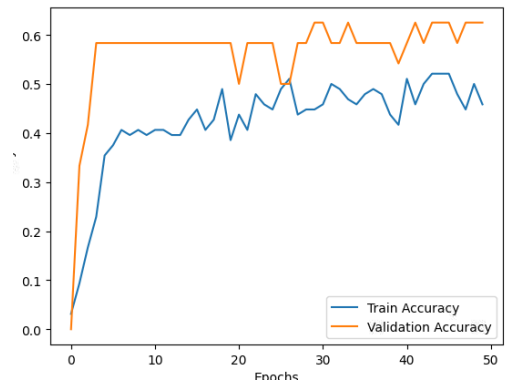
```
# Sauvegarder le modèle entraîné
model.save("modele_tensorflow_DeepL.h5")
print("Modèle sauvegardé sous 'modele_tensorflow_DeepL.h5'.")
```

```
# Sauvegarder l'encodeur des étiquettes
joblib.dump(pd.factorize([tuple(rgb) for rgb in X])[1], "label_encoder_tensorflow_DeepL.pkl")
print("Encodeur des étiquettes sauvegardé sous 'label_encoder_tensorflow_DeepL.pkl'.")
```

### 5.8 Resultat

```
Epoch 50/50
12/12 [=====] - 0s 31ms/step - loss: 1.3584 - accuracy: 0.4833
loss: 1.8538 - val_accuracy: 0.5833
Indices des classes dans les données d'entraînement : [0, 2, 4, 5, 6, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29]

Évaluation sur l'ensemble de test :
1/1 [=====] - 1s 1s/step - loss: 1.8538 - accuracy: 0.5833
Précision : 58.33%
```



## 6. Programme de tri avec le model tensorflow

Ce programme utilise un **Raspberry Pi**, des **servomoteurs**, un **capteur de couleur**, et un **modèle TensorFlow** pour **trier des M&M's** en fonction de leurs couleurs détectées

### 6.1. Initialisation et configuration

- **Raspberry Pi et servomoteurs :**
  - Deux servomoteurs sont initialisés avec des largeurs d'impulsion spécifiques pour leur fonctionnement :
    - `servo_horizontal` : contrôle les mouvements horizontaux (chargement, mesure, éjection).
    - `servo_vertical` : contrôle les mouvements verticaux (position pour chaque couleur détectée).
- **Modèle TensorFlow :**
  - Le modèle préentraîné (`modele_tensorflow_DeepL.h5`) est chargé pour identifier la couleur (classe) d'un M&M en fonction de ses valeurs RGB.
- **Table des positions (`SERVO_POSITIONS`) :**
  - Définit les positions des servomoteurs pour chaque étape (chargement, mesure, éjection) et pour chaque couleur (index de classe).

### 6.2. Fonctionnement

1. **Initialisation du capteur de couleur :**
  - Le capteur I2C est configuré pour détecter les couleurs.
  - Les données brutes (RGB) sont lues et calibrées pour corriger les variations matérielles.

#### 2. Détection de la couleur :

- Les valeurs RGB sont envoyées au modèle TensorFlow pour obtenir une prédiction.
- La prédiction renvoie un **index de classe** représentant la couleur détectée.

#### 3. Positionnement des servomoteurs :

- Selon l'index de la couleur détectée :
  - `servo_horizontal` positionne le M&M en mesure, puis en éjection.
  - `servo_vertical` déplace le M&M à la position correspondant à sa couleur.

#### 4. Triage des M&M's :

- Le programme suit une boucle infinie :
  - Le M&M est chargé en position de mesure.
  - Sa couleur est détectée.
  - Il est déplacé à la position correspondant à sa couleur ou à une position par défaut si la couleur est inconnue.

### 6.3. Points clés

- **Capteur de couleur :**
  - Lit les valeurs RGB via I2C et applique une correction (calibrage).
- **Modèle TensorFlow :**
  - Classe les couleurs détectées en utilisant un modèle préentraîné.
- **Servomoteurs :**
  - Contrôlent les mouvements pour trier les M&M's selon leurs couleurs.
- **Sécurité :**
  - Une interruption (`KeyboardInterrupt`) désactive les servomoteurs pour éviter tout dommage.

### Résumé global

Le programme automatise le tri de M&M's par couleur en utilisant un capteur de couleur pour détecter leurs valeurs RGB, un modèle TensorFlow pour classer les couleurs, et des servomoteurs pour les déplacer vers des emplacements spécifiques.

```
from gpiozero import Servo
from time import sleep
from gpiozero.pins.pigpio import PiGPIOFactory
import smbus
import time
import tensorflow as tf

# Configuration du Raspberry Pi
factory = PiGPIOFactory()

# Initialisation des servomoteurs
servo_horizontal = Servo(18, min_pulse_width=0.56/1000, max_pulse_width=1.85/1000,
pin_factory=factory)
servo_vertical = Servo(17, min_pulse_width=0.52/1000, max_pulse_width=1.7/1000,
pin_factory=factory)

# Charger le modèle TensorFlow
```

```
modele = tf.keras.models.load_model('modele_tensorflow_DeepL.h5')

# Définir les positions des servomoteurs pour chaque index de couleur
SERVO_POSITIONS = {
    "horizontal": {
        "chargement": -0.98,
        "mesure": 0,
        "ejection": 0.98
    },
    "vertical": {
        25: -0.94, # Jaune
        27: -0.7, # Orange
        15: -0.3, # Marron
        9: 0.1, # Rouge
        28: 0.1, # Rouge
        12: 0.55, # Vert
        13: 0.55, # Vert
        4: 0.95 # Bleu
    }
}

# Initialisation du bus I2C pour le capteur de couleur
bus = smbus.SMBus(1)

def afficher_indices_classes():
    print("Correspondance des indices et des positions des couleurs dans SERVO_POSITIONS :")
    for index, position in SERVO_POSITIONS["vertical"].items():
        print(f"Index : {index}, Position servo : {position}")

# Fonction : Initialiser le capteur de couleur
def init_sensor():
    bus.write_byte_data(0x39, 0x00 | 0x80, 0x03) # Activer le capteur
    print("Capteur initialisé.")

# Fonction : Lire les données de couleur depuis le capteur
def read_colors():
    bus.write_byte_data(0x39, 0x07 | 0x80, 0x00) # Configuration du gain
    time.sleep(0.5)
    data = bus.read_i2c_block_data(0x39, 0x10 | 0x80, 8) # Lecture des données
    red = data[3] * 256 + data[2]
    green = data[1] * 256 + data[0]
    blue = data[5] * 256 + data[4]

    # Calibrage des couleurs pour corriger les déviations
    red = max(0, red - 19)
    green = max(0, green - 33)
    blue = max(0, blue - 16)

    print(f"Valeurs RGB : R={red}, G={green}, B={blue}")
```



```
return red, green, blue
```

```
# Fonction : Détecter l'index de la couleur
```

```
def detect_color_index():
```

```
    red, green, blue = read_colors()
```

```
    # Préparer les données pour le modèle TensorFlow
```

```
    nouvelle_couleur = [[red, green, blue]] # Forme attendue : liste de listes
```

```
    # Faire la prédiction
```

```
    prediction = modele.predict(nouvelle_couleur)
```

```
    print(f"Probabilités de la prédiction : {prediction[0]}")
```

```
    couleur_index = tf.argmax(prediction[0]).numpy() # Index de la classe prédite
```

```
    print(f"Index de la couleur détectée : {couleur_index}")
```

```
    return couleur_index
```

```
# Fonction : Positionner les servomoteurs
```

```
def position_servo(servo, position):
```

```
    servo.value = position
```

```
    sleep(0.5)
```

```
# Fonction : Trier les M&M's
```

```
def tri_m_and_ms():
```

```
    while True:
```

```
        print("Déplacement en position de chargement...")
```

```
        position_servo(servo_horizontal, SERVO_POSITIONS["horizontal"]["chargement"])
```

```
        print("Déplacement en position de mesure...")
```

```
        position_servo(servo_horizontal, SERVO_POSITIONS["horizontal"]["mesure"])
```

```
        couleur_index = detect_color_index()
```

```
        if couleur_index in SERVO_POSITIONS["vertical"]:
```

```
            print(f"Tri pour la couleur d'index : {couleur_index}")
```

```
            position_servo(servo_vertical, SERVO_POSITIONS["vertical"][couleur_index])
```

```
            print("Éjection...")
```

```
            position_servo(servo_horizontal, SERVO_POSITIONS["horizontal"]["ejection"])
```

```
        else:
```

```
            print(f"Couleur inconnue (index {couleur_index}), éjection à une position par défaut.")
```

```
            position_servo(servo_vertical, 0) # Position par défaut ou d'échec
```

```
        sleep(1) # Pause avant la prochaine itération
```

```
# Programme principal
```

```
if __name__ == "__main__":
```

```
    try:
```

```
        print("Initialisation du capteur...")
```

```
        init_sensor()
```

```
        afficher_indices_classes()
```

```
        sleep(10) # Pause avant la prochaine itération
```

```
print("Début du tri...")
tri_m_and_ms()
except KeyboardInterrupt:
    print("Programme arrêté par l'utilisateur.")
finally:
    # Désactiver les servomoteurs
    servo_horizontal.value = None
    servo_vertical.value = None
    print("Servos désactivés.")
```

Resultat obtenu il y a trois M&Ms bleu avec les verts et deux rouges avec les oranges



# MERCI DE VOTRE ATTENTION

