Rapport technique
Projet Ambilight

Réalisé par Flora Silberzan, Titouan Delforge, Sapna Hassanaly et Ilian Ellafi

https://www.domo-blog.fr/test-de-dreamscreen-solution-de-lambilight-ecrans/

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech
LEnsE Laboratoire d'Enseignement Expérimental

I. Introduction

Notre projet consiste à modéliser un éclairage ambilight. Il s'agit d'un dispositif d'éclairage situé sur le bord de l'écran qui, à l'aide de LEDs, prolonge l'image projetée sur l'écran. Ceci permet alors de créer une ambiance plus immersive et des effets lumineux. Ce dispositif existe déjà sur des téléviseurs où l'image affichée est directement traitée. Cependant, lors de notre projet, nous avons décidé de capter l'image projetée par n'importe quel type d'écran à l'aide d'une webcam. Ensuite, le système traite l'image captée sur python à l'aide d'une carte Raspberry Pi 3 et enfin transmet les commandes au LEDs pour les éclairer de la bonne couleur. Ainsi l'objectif est de récupérer l'image affichée par un écran à intervalle régulier à l'aide d'une caméra, et éclairer les 4 coins de l'écran de la couleur moyenne du bord de l'image grâce aux bancs de LEDs.

II. Découpage fonctionnelle

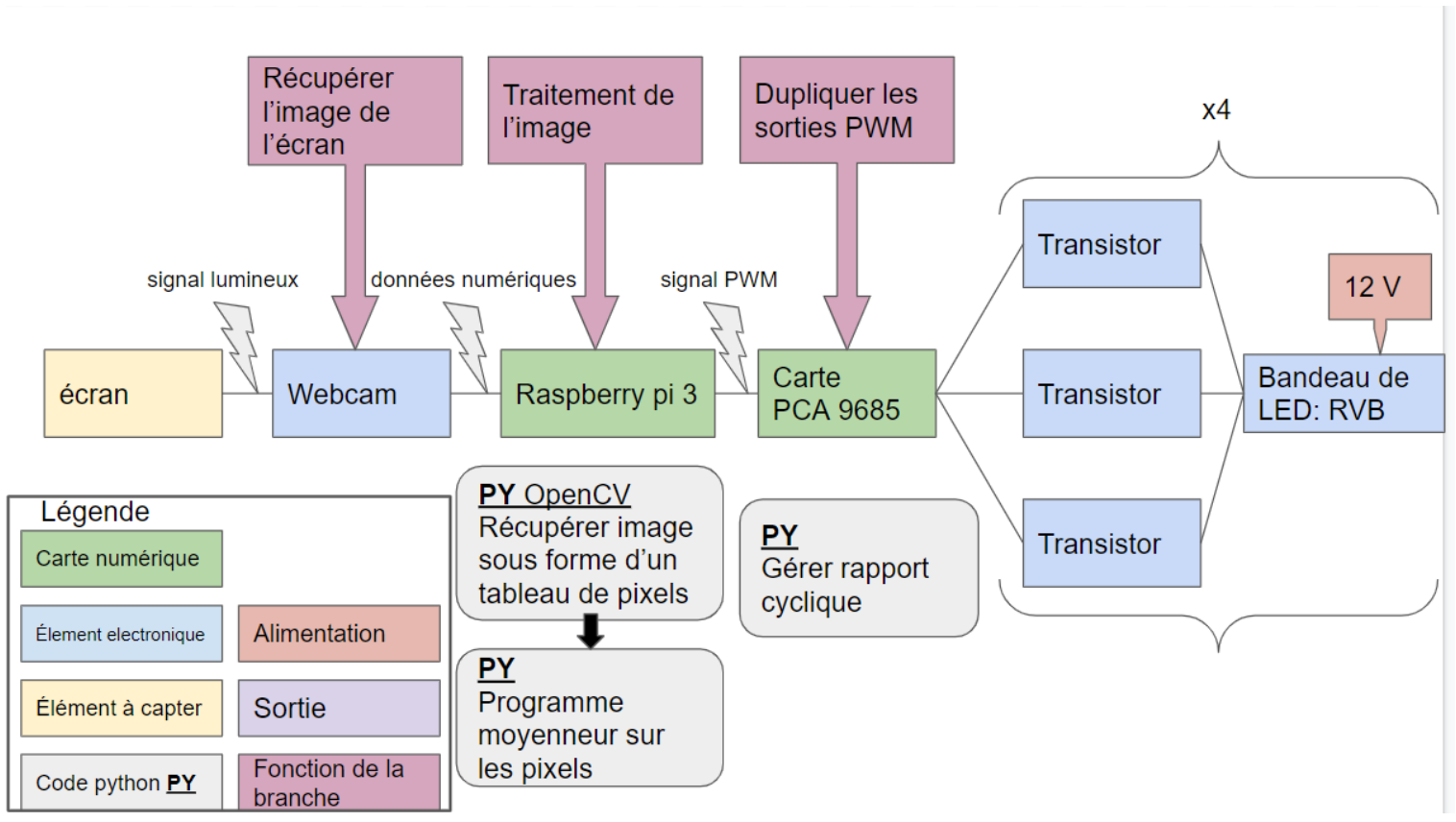

Figure 1: Schéma bloc du système étudié

Principe de fonctionnement du système :

L'écran est filmé par la caméra, la carte Raspberry Pi récupère l'image sous forme d'un tableau de pixels, effectue un traitement d'image puis génère des signaux PWM qui seront envoyés à quatre bancs de LEDs par l'intermédiaire d'une carte de duplication (PCA9685). Chaque banc de LEDs est relié à trois transistors et alimenté en 12V continu.

La carte Raspberry

Figure 2: Image de la carte Raspberry Pi 3 https://www.ldlc.com/fiche/PB00205573.html

Un des objectifs de notre projet est d'automatiser tout notre installation : que les LEDs suivent en temps réel l'image captée par la caméra.

Au début de ce projet nous avions quelques bases sur l'utilisation et la programmation de microcontrôleurs (carte Nucléo). Cependant la récupération des données par la caméra, le traitement de ces informations, ainsi que la transmission jusqu'aux LEDs demandait trop de mémoire pour un microcontrôleur classique…On s'est donc tourné vers l'utilisation d'une carte Raspberry bien plus puissante, carrément sous un système d'exploitation Linux. Le plus intéressant pour nous avec cette nouvelle carte est la RAM, en effet, on passe de 2 kB pour une simple carte nucléo à environ 1GB pour la Raspberry. Ainsi plus proche d'un mini-ordinateur, cette carte Raspberry nous a permis de traiter la quantité d'informations nécessaire pour notre projet.

III. Réalisation du prototype

A. Récupération de la capture de la caméra

Nous avons tout d'abord réussi à récupérer une image sur python via la bibliothèque OpenCv grâce à notre fonction récupérationphoto. On a donc pu récupérer une liste de taille 480x640x3 correspondant à l'image captée par la caméra. On a donc les 3 couleurs Rouge Vert Bleu représenté dans ces tableaux pour chaque pixels dans l'ordre inverse : Ainsi un pixel contiendra en premier le pixel bleu puis celui vert puis celui rouge.

Au niveau du fonctionnement de la fonction, la caméra est détectée automatiquement, puis on capture une image avec cap.read. Pour visualiser cette image on utilise la commande cv.imshow. Les commandes suivantes permettent de supprimer de l'écran l'image affichée en appuyant sur q sur le clavier. On "libère" finalement l'image prise de la caméra avec cap.release() et on retourne le tableau de l'image (cf annexe).

Traitement image

Après avoir récupéré le tableau de pixels de dimension (480x640x3), la prochaine étape du projet consiste à traiter le tableau de pixels à l'aide du module numpy. Comme nous avons décidé de fixer les LEDs sur les coins de l'écran seuls les coins de l'image captée par la caméra seront traités (zones en rouge sur le schéma ci-dessous).

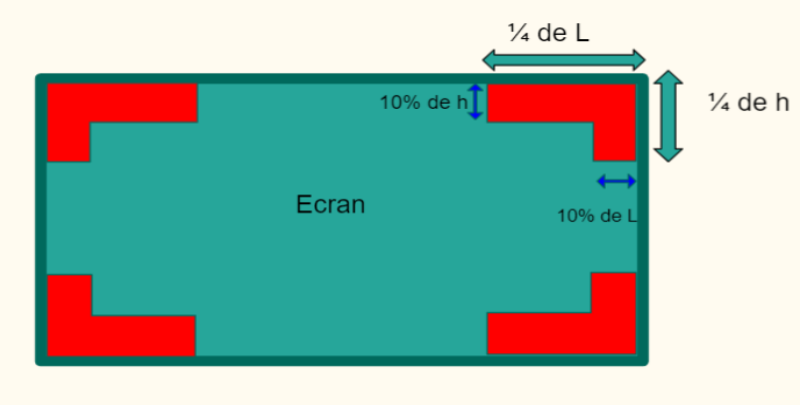

Figure 3: Répartition des zones de l'image qui vont être traitées (en rouge parties traitées, en bleu parties non traitées)

Le traitement d'image consiste à moyenner séparément les valeurs de pixels sur chaque zone en rouge du schéma pour chaque couleur. La fonction TimageOpti1 (cf annexe) renvoie donc un tableau de 12 valeurs qui permettront de piloter la couleur des LEDs : une valeur pour chaque couleur (rouge, vert, bleu) et pour chacune des 4 zones.

B. Pilotage des LEDs

Les bandeaux de LEDs

Les bandeaux de LEDs que nous avons utilisés sont constitués de LEDs trichromes (RVB), notre but est de pouvoir contrôler chaque couleur séparément. Ils sont alimentés par une tension de 12V.

Figure 4: Bandeau de LEDs utilisé (Source: LENSE)

Carte PCA 9685

Pour modifier la luminosité d'une LEDs ou changer la couleur d'un bandeau de LEDs RVB, il est nécessaire de pouvoir faire varier la tension en entrée de notre bandeau. Pour réaliser cette variation de tension à partir de notre sortie digitale de la carte Raspberry, et ainsi générer un signal analogique, on a utilisé un signal à rapport cyclique variable (PWM).

On peut alors faire varier le rapport cyclique de chacune des LEDs (R, V, B) et ainsi en fonction de toutes les luminosités de chacune des LEDs obtenir les différentes couleurs souhaitées sur le bandeau.

Cependant la carte Raspberry ne comporte que quelques sorties PWM, et comme nous venons de le dire, on veut faire varier la luminosité de chacune des LEDs qui composent les bandeaux (Soit 3*nombre de bandeaux). On a donc dû dupliquer une des sorties PWM de la carte par une carte PCA 9685 et ainsi par le biais de bibliothèque `smbus` sur Python, on est capable de contrôler indépendamment chacune des 16 sorties PWM de la carte.

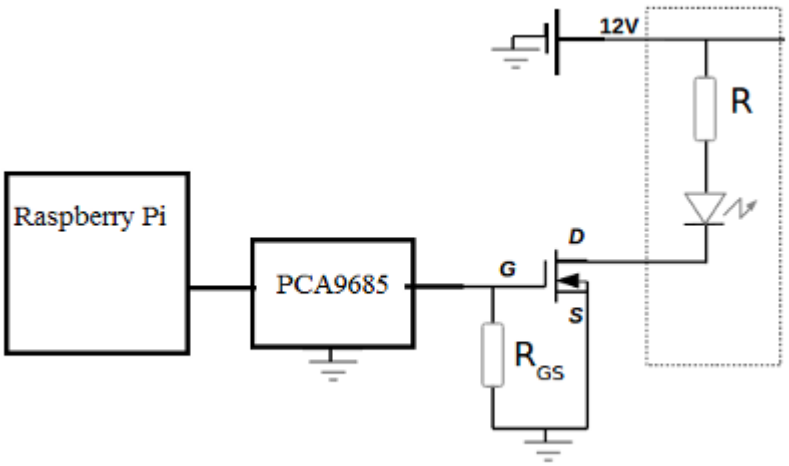

Figure 5: Schéma du montage électronique

Transistors

Les bandeaux de LEDs que nous utilisons sont alimentés sous du 12V, cependant la carte ne fournit pas autant de courant. Pour amplifier le courant on a utilisé des transistors, en particulier MOSFET qui est commandé en tension. (cf Figure 5)

Les conditions pour le choix du transistor était qu'il fallait que :

- La tension devait être supérieure à la tension d'alimentation du bandeau (donc ici 12V).
- Le courant devait être supérieur au courant maximal consommé par chacune des voies.
- La tension minimale de commande devait être inférieure à la tension de sortie de la carte Raspberry.

D'après toutes ces caractéristiques, on a choisi le transistor BS170.

Pilotage des LEDs

Après avoir effectué le montage des LEDs, une des difficultés a été de comprendre comment était codé le PWM et de trouver les commandes et adresses des sorties de la carte PCA9685 dans la documentation. Après avoir importé la bibliothèque `smbus`, on initie le code ainsi :

```
DEVICE_BUS = 1
DEVICE_ADDR =0x40
bus =smbus.SMBus(DEVICE_BUS)
```

```
bus.write_byte_data(DEVICE_ADDR,
0x00, 0x21)
```

Chaque couleur (rouge, vert, bleu) de chaque banc de LEDs est pilotée par un signal PWM dont la durée de l'impulsion est codée sur 12 bits. On commande séparément les 4 bits de poids forts, qui peuvent prendre des valeurs entre 0 et 15, et les 8 bits de poids faibles qui peuvent prendre des valeurs entre 0 et 255.

Pour chaque couleur, on utilise le code suivant :

```
#deuxpremières lignes pour le début de l'impulsion
bus.write_byte_data(DEVICE_ADDR,0x06, 0x0)
bus.write_byte_data(DEVICE_ADDR,0x07, 0x0)
#puis pour la fin de l'impulsion
bus.write_byte_data(DEVICE_ADDR,0x08, M[i+12])#poids faibles
bus.write_byte_data(DEVICE_ADDR,0x09, M[i]) #poids forts
```

Où M est le tableau contenant les valeurs des bits de poids forts dans les 12 premières cases et les bits de poids faibles dans les douze suivantes

On peut donc désormais coder 2^{12} = 4096 intensités lumineuses pour chaque couleur, soit 2^{36}nuances de couleur par banc de LEDs.

Circuit imprimé

Un des intérêts de ce projet est de pouvoir rendre la technologie Ambilight plus compacte, de ne pas obligatoirement posséder la télévision adéquate et ainsi d'être possiblement transportable. Ainsi pour éviter de refaire tous les branchements à chaque utilisation, on a opté pour une carte imprimée qui regroupe tous les branchements en une seule pièce.

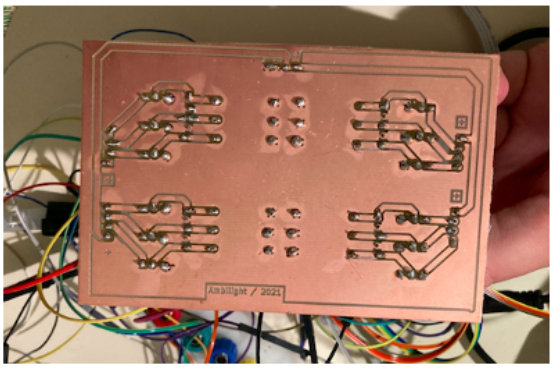

Figure 6: Photographie de du circuit imprimé

C. Automatisation du code

Le procédé de récupération de l'image n'étant pas automatique, on a voulu lui donner cette capacité. Ainsi le but était de faire tourner le code de façon permanente et que les LEDs se mettent à jour en continu selon l'image captée par la caméra. Ainsi on a créé la fonction de traitement permanent Tinfini() (cf annexe) qui donne ce résultat. On a donc fait une boucle infinie while(1). On reprend les commandes utilisées dans récupération photo, à savoir VideoCapture pour lancer la caméra et cap.read() pour capturer une image. On n'utilise plus le bouton q pour quitter l'image puisqu'on n'affiche pas cette image.

Dans les cas exceptionnels pour sortir de la boucle infini, on a programmé avec les commandes try et except une sortie de la boucle : En utilisant la commande Keybord Interrupt, cela permet, lorsqu'on appuie sur Ctrl + C, d'exécuter exceptionnellement la commande break, qui arrête instantanément la boucle while infini.

IV. Tests et validation du projet

Nous avons réussi à faire une boucle infinie qui permet d'exécuter le code automatiquement. Nous avons mesuré pour une boucle un temps d'exécution de 50 ms. Pour pouvoir améliorer ce temps d'exécution il faudrait retravailler les fonctions utilisées, notamment la fonction de traitement d'image pour diminuer leur complexité, mais surtout utiliser une carte encore plus puissante et plus adaptée pour le calcul graphique.

Notre système fonctionne bien pour les 3 couleurs primaires rouge vert bleue : la couleur prise par les LEDS est proche de la couleur affichée sur l'écran.

Figure 7: Photographie du prototype en fonctionnement

Cependant le système ne fonctionne pas très bien pour les couleurs plus complexes qui sont un mélanges des trois couleurs de pixels comme le orange et le jaune. En effet, si il existe une toute petite représentation de la couleur bleue, elle prend le dessus et dénature la couleur des LEDs.

V. Comprendre les étapes de réalisation

A. Planning

Séance 1 : jeudi 21 janvier	Trouver le sujet personnel, commencer le cahier des charges. Répartition des tâches : Flora et Titouan sur le câblage et gestion des LEDs et Sapna et Ilian sur le code et la gestion de la caméra. Découverte de la carte Raspberry Pi 3.
Séance 2 : mercredi 27 janvier	Découverte de la carte PCA9685, lecture de documentation technique. Découverte du module OpenCV pour récupérer une image via la webcam et du format de l'image colorée (RVB/ matrice). Début de la fonction traitement image.
Séance 3 : mercredi 24 février	Ecriture du code pour allumer et contrôler un banc de LED (seulement poids forts) montage électrique pour un banc de LED : fonctionnel. Non fonctionnel pour 4 bancs de LEDs. Demande de circuit imprimé. Fin de la fonction traitement image et test de la fonction. Pareil pour la fonction récupération image et configuration de la webcam.
Séance 4 : mercredi 3 mars	Montage fonctionnel pour quatre bancs de leds séparés. Mise en commun des codes. élaboration d'un code prenant en compte les poids faibles. Test des fonctions avec les bancs de LEDs. Automatisation de la fonction récupération image. Prise de conscience du problème de colorimétrie et du temps d'exécution.
Séance 5 : lundi 22 mars	Soudure du circuit imprimé. Montage avec la carte Raspberry et les quatre bancs de LEDs. Recherche sur les temps d'exposition de la caméra (impossible sur logitech). Création de fonction de calibrage de colorimétrie (pas optimale). Optimisation du temps d'exécution de la fonction traitement image. Un test est nécessaire.
Séance 6 : lundi 29 mars	Test et correction sur l'optimisation de la fonction de traitement. Réflexion sur les fonctions de colorimétrie tous ensemble. Équiper l'écran et câbler l'ensemble. Test du système complet avec un diaporama d'images colorées sélectionnées et avec une vidéo.

Séance 7 : lundi 3 mai	Finalisation des fonctions restantes : débuggage de la fonction d'automatisation. Finaliser le projet. Derniers tests. Réflexion et création du rendu final pour la présentation finale.
Séance 8 : lundi 17 mai	Derniers réglages et derniers tests. Réflexions sur les livrables finaux. Présentation finale.

B. Difficultés rencontrées

Une des difficultés majeures rencontrées lors de ce projet est le problème de colorimétrie. En effet, lorsque la caméra filme la l'écran, il peut y avoir un décalage entre la couleur projetée par l'écran et la couleur captée par la caméra. On peut voir un exemple sur l'image suivante :

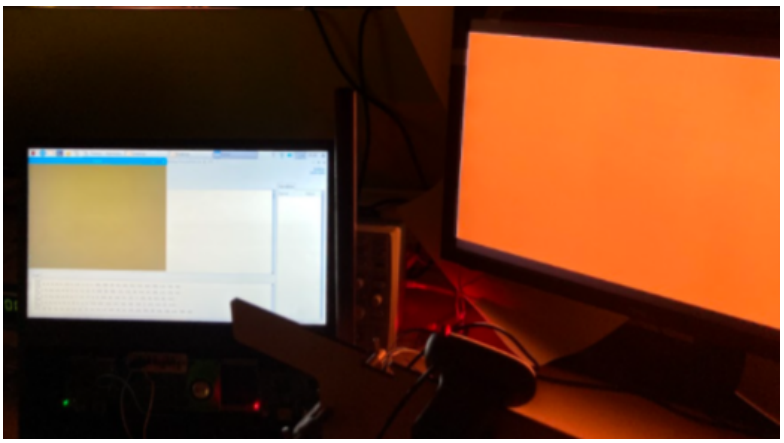

Figure 8 : Illustration du problème de colorimétrie: à droite couleur originale affichée à l'écran (orange) et à gauche couleur captée par la caméra (plus terne presque gris).

Si ce décalage persiste, cela induit une erreur sur la couleur prise par les LEDs. Pour corriger ce problème, on a essayé d'appliquer la méthode de la balance de blancs. Lorsque l'on filme une image blanche, la caméra est censée capter dans l'idéal le pixel 255 pour les 3 couleurs. Or dans la pratique, ce n'est pas le cas. La méthode de la balance des blancs consiste à multiplier pour chaque couleur le pixel par la valeur captée lorsque l'image est blanche. Malheureusement, la fonction de colorimétrie ne fonctionne pas très bien: si certaines couleurs prises par les LEDs sont beaucoup plus proches de celle affichée sur l'écran (bleu/ rouge/ vert) , pour d'autres couleurs la différence augmente: la fonction ne corrigeait pas le décalage sur les couleurs qui n'apparaissaient déjà pas bien sur les LEDs. Nous avons donc choisi au final de ne pas mettre la fonction de colorimétrie.

Un deuxième problème était que nous n'avions pas accès au software de la caméra. Il n'était pas possible de régler le temps d'exposition ou de faire un étalonnage du blanc pour essayer de compenser le problème de colorimétrie.

Par ailleurs, lors de notre codage, nous devions veiller à ce que toutes les valeurs de pixels soient entières et comprises entre 0 et 255, pour éviter un dépassement d'octet ou un problème lors de l'exécution.

Pour le pilotage des LEDs, nous avions au départ décidé de ne pas prendre en compte les poids faibles, et que les 16 nuances des poids forts fournissaient assez de nuances de couleur. Mais nous nous sommes rendus compte que cela posait des problèmes pour les couleurs chaudes, qui ne rendaient pas bien à cause de la présence de bleu. Nous avons donc créé une fonction (divisé16) pour les ajouter dans le tableau des moyennes.

C. Analyse du travail d'équipe

La répartition du travail s'est faite naturellement au début du projet. Ilian et Sapna se sont occupés de la gestion de la caméra et du traitement de l'image (partie codage). Titouan et Flora se sont occupés du câblage et du pilotage des LEDs (partie électronique) . On a réussi à travailler au même rythme entre les deux binômes. Il aurait peut-être été plus utile de plus mélanger les tâches pour que chacun puisse travailler à la fois sur la partie électrique et sur la partie du code.

VI. Conclusion

Le prototype de ce projet Ambilight a finalement abouti au bout de 7 séances. Cependant nous avons rencontré certaines difficultés (comme la colorimétrie, certaines contraintes dues à la caméra ou encore le pilotage des LEDs) que nous avons réussies à surmonter pour présenter un résultat satisfaisant.
Le système pourrait également être perfectible en améliorant le temps de calcul de notre programme. Une idée pour être plus rapide est d'utiliser une carte graphique, mais cela augmenterait considérablement le prix de ce projet… En effet, actuellement, le projet à l'avantage d'être bien moins cher que la télévision Ambilight proposée par Philips, et ainsi abordable pour des étudiants !

Annexe:

```
import cv2 as cv
import numpy as np
import math
import statistics as stat
import time
import smbus
import time
from datetime import datetime

def recuperationphoto():  ## récupérer une photo de la webcam

      #lance la caméra
    if cap.isOpened() :
        ret,frame = cap.read()     #capture une image de la caméra
    #cv.imshow('frame',frame)   #montre l'image capturé

    #if cv.waitKey(0) & 0xFF == ord('q'):  #Mettre la suite en commentaire si on ne
veut pas cliquer sur q pour fermer la fenêtre de l'image.
      #  cv.destroyAllWindows() #supprime la fenêtre de l'image
        cap.release()
    return frame    #renvoie l'image


def TimageOpti1(Image):
    (ligne,colonne,pxl)=np.shape(Image)
    newl=ligne//2
    newc=colonne//2
    L4=ligne//4
    C4=colonne//4
    lstop=round(ligne*0.10)  #round permet de prendre l'arrondi
    cstop=round(colonne*0.10)
    M=[]

#cote haut gauche
    for p in range (3):  # pour traiter chaque couleur un par un
        L1=np.mean(Image[0:lstop,0:C4,p])
        L2= np.mean(Image[lstop:L4,0:cstop,p])
        M.append(int(round(np.mean([L1,L2]))))  ## M est peut etre un flottant (le
convertir en entier pour le PWM

#coté haut droit
    for p in range (3):  # pour traiter chaque couleur un par un
        L1=np.mean(Image[0:lstop,colonne-C4:colonne,p])
        L2= np.mean(Image[lstop:L4,(colonne-cstop):colonne,p])
        M.append(int(round(np.mean([L1,L2]))))  ## M est peut etre un flottant (le
convertir en entier pour le PWM

#cote bas gauche
    for p in range (3):  # pour traiter chaque couleur un par un
        L1=np.mean(Image[ligne-L4:ligne,0:cstop,p])
        L2= np.mean(Image[ligne-lstop:ligne,cstop: C4,p])
        M.append(int(round(np.mean([L1,L2]))))  ## M est peut etre un flottant (le
convertir en entier pour le PWM

#cote bas droit
    for p in range (3):  # pour traiter chaque couleur un par un
```

```
        L1=np.mean(Image[ligne-L4:ligne,colonne-cstop:colonne,p])
        L2= np.mean(Image[ligne-lstop:ligne,colonne-C4:colonne-cstop,p])
        M.append(int(round(np.mean([L1,L2]))))  ## M est peut etre un flottant (le
convertir en entier pour le PWM

    return M


def calibrage(M): # méthode de la balance des blancs
 #1,78 est le coefficient choisi pour que les trois couleurs soient le plus proche
de 255 lorsque l'image est blanche.

    M[0:,0:,0]=255/(1.78*131)*M[0:,0:,0]
    M[0:,0:,1]=255/(1.5*142)*M[0:,0:,1]
    M[0:,0:,2]=255/(1.78*134)*M[0:,0:,2]
    M=np.around(M)
    return M

def divisé16(M):  #permet de mettre les valeurs moyenne entre 0 et 15
    R = []
    for i in range(12):
        M.append((M[i]%16)*16)
        M[i]=M[i]//16
    return M


def affichage(M) : #commande la luminosité et la couleur des LEDs

    #print('test')
    DEVICE_BUS = 1
    DEVICE_ADDR = 0x40
    bus = smbus.SMBus(DEVICE_BUS)
    bus.write_byte_data(DEVICE_ADDR, 0x00, 0x21)
    time.sleep(.01)
    #LED haut gauche
    #Bleu
    bus.write_byte_data(DEVICE_ADDR, 0x06, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x07, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x08, M[12])
    bus.write_byte_data(DEVICE_ADDR, 0x09, M[0])
    #Vert
    bus.write_byte_data(DEVICE_ADDR, 0x0A, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x0B, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x0C, M[13])
    bus.write_byte_data(DEVICE_ADDR, 0x0D, M[1])
    #Rouge
    bus.write_byte_data(DEVICE_ADDR, 0x0E, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x0F, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x10, M[14])
    bus.write_byte_data(DEVICE_ADDR, 0x11,M[2])

    #LED haut droit
    #Bleu
    bus.write_byte_data(DEVICE_ADDR, 0x16, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x17, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x18, M[15])
    bus.write_byte_data(DEVICE_ADDR, 0x19, M[3])
    #Vert
    bus.write_byte_data(DEVICE_ADDR, 0x1A, 0x0)
```

```
    bus.write_byte_data(DEVICE_ADDR, 0x1B, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x1C, M[16])
    bus.write_byte_data(DEVICE_ADDR, 0x1D, M[4])
    #Rouge
    bus.write_byte_data(DEVICE_ADDR, 0x1E, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x1F, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x20, M[17])
    bus.write_byte_data(DEVICE_ADDR, 0x21, M[5])


    #LED bas gauche
    #Bleu
    bus.write_byte_data(DEVICE_ADDR, 0x26, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x27, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x28, M[18])
    bus.write_byte_data(DEVICE_ADDR, 0x29, M[6])
    #Vert
    bus.write_byte_data(DEVICE_ADDR, 0x2A, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x2B, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x2C, M[19])
    bus.write_byte_data(DEVICE_ADDR, 0x2D, M[7])
    #Rouge
    bus.write_byte_data(DEVICE_ADDR, 0x2E, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x2F, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x30, M[20])
    bus.write_byte_data(DEVICE_ADDR, 0x31, M[8])


    #LED bas droit
    #Bleu
    bus.write_byte_data(DEVICE_ADDR, 0x36, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x37, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x38, M[21])
    bus.write_byte_data(DEVICE_ADDR, 0x39, M[9])
    #Vert
    bus.write_byte_data(DEVICE_ADDR, 0x3A, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x3B, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x3C, M[22])
    bus.write_byte_data(DEVICE_ADDR, 0x3D, M[10])
    #Rouge
    bus.write_byte_data(DEVICE_ADDR, 0x3E, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x3F, 0x0)
    bus.write_byte_data(DEVICE_ADDR, 0x40, M[23])
    bus.write_byte_data(DEVICE_ADDR, 0x41, M[11])
    time.sleep(0.5)

def Tinfini() : #traite et envoie les informations aux LEDs de façon continue !
    while(1):
        try :
            cap=cv.VideoCapture(cv.CAP_V4L2)
            #récupérer l'image en continue et appliquer les fonctions précédentes.
            #tps1=time.clock()
            if cap.isOpened() :
                ret,frame = cap.read()

            #print(frame)
            #M=calibrage(frame)
            cap.release()
```

```
            M=TimageOpti1(frame)
            M = divisé16(M)
            affichage(M)
            print(M)

            cv.destroyAllWindows()
        except KeyboardInterrupt :  # faire contrôle C pour interrompre la fonction
            break
        #tps2=time.clock()
        #print(tps2-tps1)




##Fonction annexe## Pour faire la méthode de la balance des blancs.

def moyennebalanceblanc(Image): #mesurer la moyenne des pixels pour la méthode de
la balance des blancs
    (ligne,colonne,pxl)=np.shape(Image)
    Lbleu=[]
    Lrouge=[]
    Lvert=[]

    for i in range (ligne):
        for j in range (colonne):
                Lbleu.append(Image[i,j,0])
                Lvert.append(Image[i,j,1])
                Lrouge.append(Image[i,j,2])
    Mrouge=stat.mean(Lrouge)  # moyenne de la valeur des pixels rouge d'une image
blanche
    Mbleu=stat.mean(Lbleu)  # moyenne de la valeur des pixels bleus d'une image
blanche
    Mvert=stat.mean(Lvert)   # moyenne de la valeur des pixels verts d'une image
blanche
    return Mbleu,Mvert,Mrouge


#im=cv.imread("imtest.png")
#print(moyennebalanceblanc(a))
#print(im)
#print(calibrage(im))

#moyennehexa()


Tinfini()
```