Rapport technique

Projet Ambilight

Taha El Berry, Alban Leleux, Flore Loiseau, Nathan Mouligneaux

Comprendre le projet

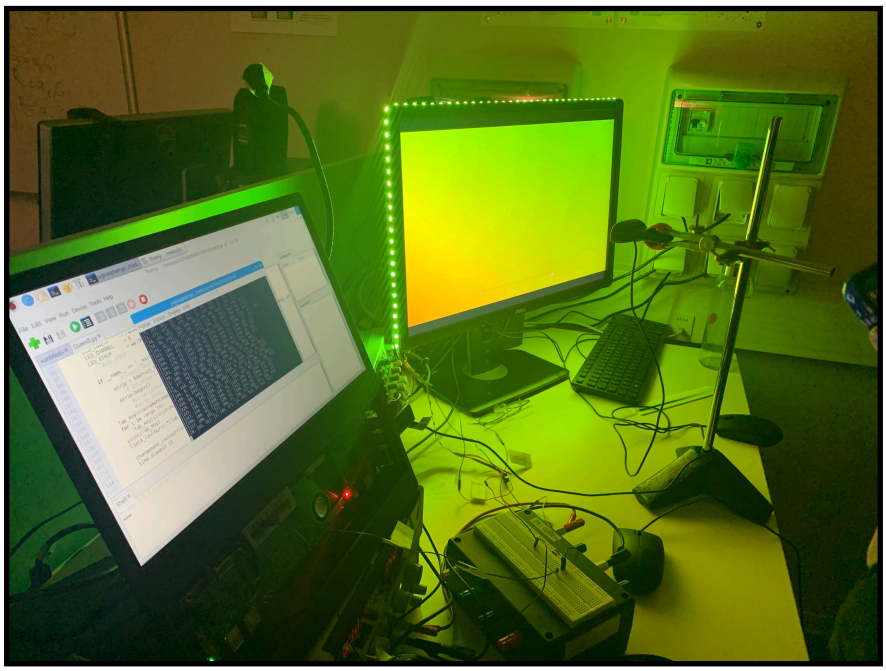

Figure 1 : Photographie de notre système Ambilight en cours de fonctionnement.

1. Introduction

Le projet Ambilight permet de prolonger les couleurs des bords d'un écran afin d'approfondir l'immersion du visionnage. Il dispose d'un dispositif d'éclairage, normalement déjà intégré sur le téléviseur, qui s'illumine spontanément selon les images projetées sur l'écran. Notre projet utilise une webcam centrée sur un écran d'ordinateur, faisant l'acquisition de l'image projetée. Les couleurs des pixels de quatre zones, correspondant aux quatre coins de l'image capturée, sont moyennées et traitées grâce à un programme Python sur une carte Raspberry qui transmet les commandes au bandeau de LEDS installé sur les bords de l'écran. Ainsi, le but de ce

projet est d'éclairer à intervalles réguliers les coins de l'écran selon les couleurs affichées sur l'ordinateur.

2. Découpage fonctionnel (Graphique synthétique des diverses fonctions réalisées / Descriptif des fonctionnalités)

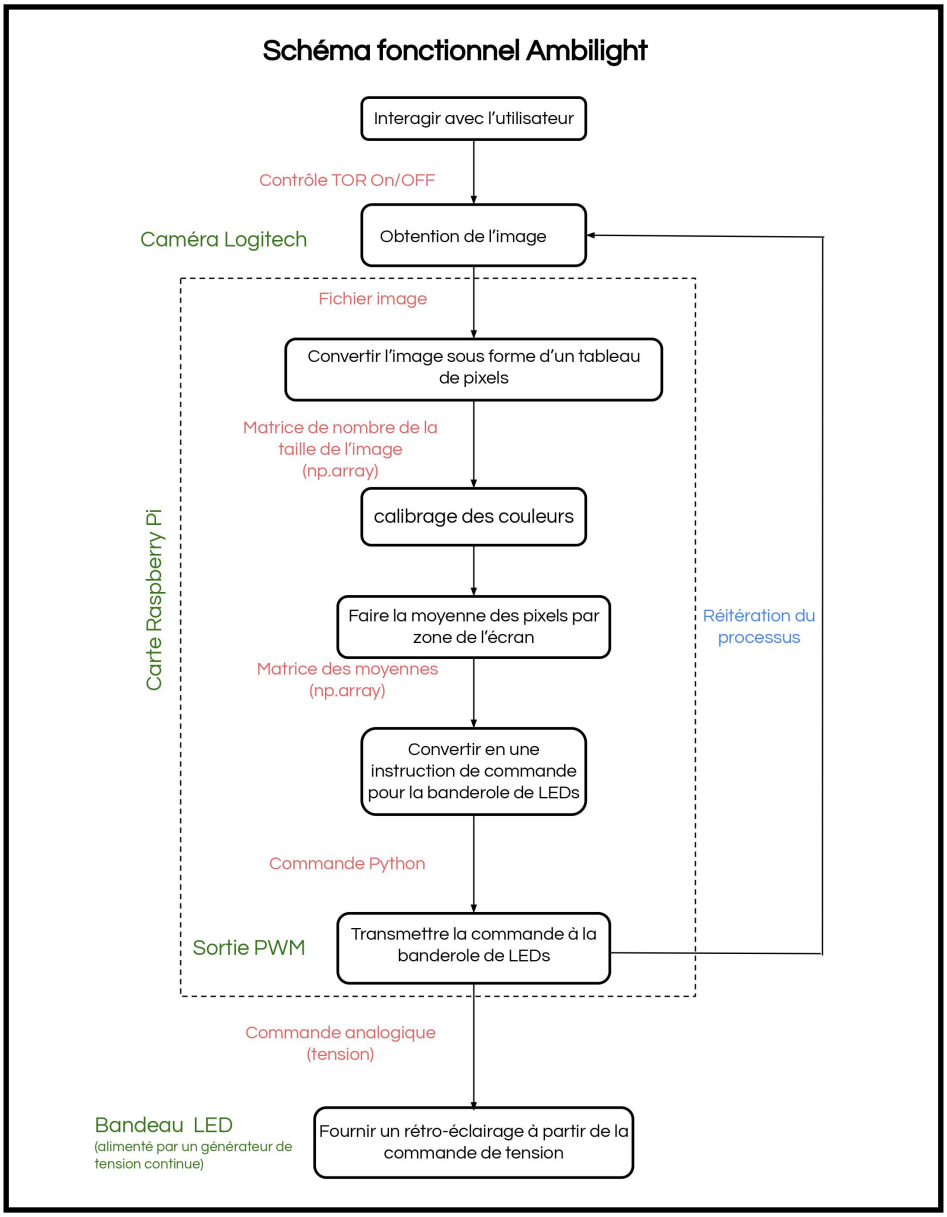

Figure 2 : schéma fonctionnel du système étudié.

Matériel utilisé :

- Webcam LogiTech
- Carte RaspberryPi 3
- Bandeau de LEDs WS2812B
- Fils
- Breadboard
- Générateur de tension continue

Réaliser le prototype (fonctionnalités)

3. Principe de fonctionnement

La caméra fait l'acquisition de l'image affichée sur l'écran devant laquelle elle est placée. L'image acquise est récupérée sous la forme d'un tableau de pixels, comportant chacun les informations des couleurs (RVB), et est traitée par la carte RaspberryPi, sous un système d'exploitation Linux. Le programme Python installé sur celle-ci traite l'image et moyenne les zones des bords (divisées en quatre zones correspondant aux quatre coins) puis commande en temps réel, à intervalles réguliers, le bandeau de LEDs selon ces quatre zones.

3.1 Récupération / Acquisition de l'image

La fonction cv.VideoCapture lance la caméra, puis cap.read capture l'image.

L'image est récupérée via la fonction python recuperationphoto qui mobilise des éléments de la bibliothèque OpenCv. Elle est désormais représentée par une matrice de triplets dont les valeurs correspondent aux pixels utilisés (dans l'ordre bleu/vert/ rouge).

3.2 Traitement de l'image par la carte RaspberryPi

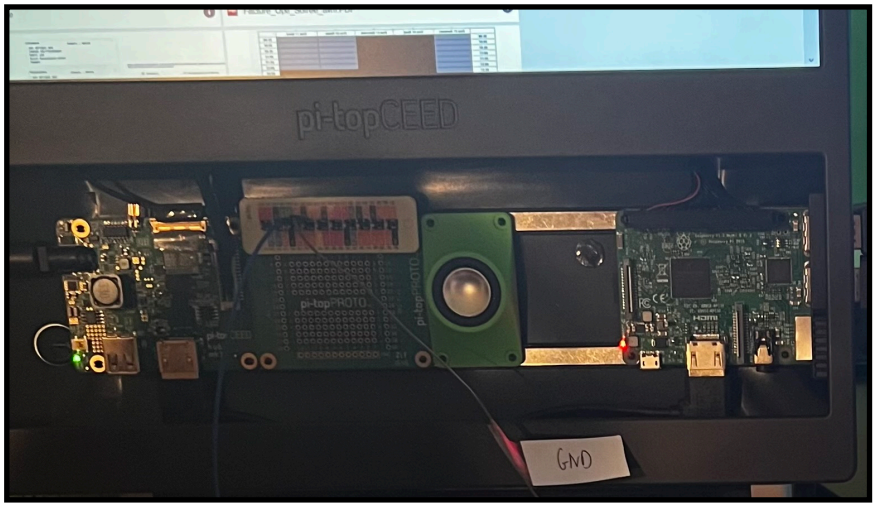

Figure 3 : Photographie de notre carte Raspberry Pi

La couleur acquise par la caméra est calibrée par une fonction permettant de corriger la couleur. En effet, l'image affichée par la caméra initialement n'étant pas fidèle aux couleurs réelles, il est nécessaire de calibrer les valeurs des pixels afin que celles-ci soient plus pertinentes par rapport à ce qui s'affiche à l'écran.

La matrice obtenue est moyennée selon quatre zones (coin haut droit, coin haut gauche, coin bas droit, coin bas gauche). Finalement, on obtient une liste des moyennes des valeurs des pixels de la matrice initiale, composée de quatre triplets.

Figure 4 : Modèle de banderole LED utilisé, WS2812B
Source : https://gsmserver.com/

3.3 Pilotage des LEDs

Les LEDs sont alimentées par un générateur annexe qui fournit une tension de 3 V. La carte RasberryPi commande directement le bandeau de LEDs via le PIN 18.

Afin de piloter les LEDs au moyen de la carte Raspberry, il est nécessaire de fournir une commande sous une forme appropriée qui puisse être interprétée par la banderole LED. À partir de la liste à 4 triplets précédente, notre code python crée une liste de 110 triplets RGB indexés chacun par la position de la LED correspondante sur la banderole et à laquelle le triplet est destiné. Cette liste sert ensuite d'argument à la fonction `SetPixelColor`, issue de la bibliothèque OpenCv et qui permet d'envoyer la commande aux LEDs.

Finalement, le programme doit être rafraîchi afin que les couleurs des LEDs s'actualisent et suivent les couleurs affichées sur l'écran, qui changent lors du visionnage d'une vidéo, par exemple. Afin de réaliser ce rafraîchissement, nous avons utilisé une boucle while, dont on contrôle la fréquence de parcours en utilisant la fonction `time.sleep` qui retarde la boucle de la durée souhaitée.

4. Algorithmes

cf. Annexe

Valider / Caractériser le système final

5. Test et validation

Le projet a été testé sur des vidéos trouvées sur internet, mais, étant donné la longueur du temps d'acquisition, le test à l'aide de vidéos montrant des plans fixes de couleurs unies ou quadrillées a été préféré.

Le principal problème rencontré était le calibrage des couleurs. Finalement, en supposant une conversion linéaire, nous avons utilisé des matrices de changement de base afin d'améliorer les valeurs moyennées finales des pixels. Le résultat final est satisfaisant, à part le bleu, qui apparaît plutôt turquoise.

6. Planning

Séances	Travail réalisé
1) 31/01/2022	Découverte du projet Rédaction de la fiche descriptive et du scénario d'image Câblage global pour se familiariser avec la carte Raspberry Comptage du nombre de LEDs sur le bandeau
2) 09/02/2022	Rédaction du livrable intermédiaire (répartition des tâches, découpage fonctionnel du système à développer) Câblage pour relier les bandeaux de LEDs à la carte Raspberry, compréhension du fonctionnement du bandeau
3) 1er/03/2022	Codage acquisition de l'image via OpenCv, Début de la fonction de traitement de l'image (moyennage)
4) 08/03/2022	Fin de la fonction d'acquisition Fonction de pilotage des LEDs Premier test à l'aide du prototype de fonction d'acquisition Problème de colorimétrie rencontré : calibrage des couleurs peu satisfaisant Recherche d'une solution selon la méthode de balance des blancs Problème de rafraîchissement : rafraîchissement trop lent

5) 13/04/2022	Automatisation de la fonction d'acquisition Optimisation de la fonction du traitement d'image Problème de calibrage
6) 20/04/2022	Réussite du rafraîchissement Encore problème de colorimétrie
7) 17/05/2022	Résolution problème de colorimétrie (résultat satisfaisant) Rédaction livrable final

Comprendre les étapes de réalisation / choix

7. Etapes de réalisation

7.1 Difficultés rencontrées

Tout d'abord, il a fallu comprendre le fonctionnement du bandeau des LEDs et leur branchement à la carte Raspberry, qu'il fallait réinstaller à chaque séance.

Ensuite, chaque lancement du programme nécessitait d'utiliser l'invité de commande du système d'exploitation , et de formuler les bonnes commandes dont la syntaxe était compliquée à retenir.

Les principales étapes qui ont causé problème étaient la colorimétrie et le rafraîchissement. Pour la colorimétrie, le problème venait de la caméra, qui ne perçoit pas bien les couleurs. En effet, le bleu, par exemple, apparaît grisâtre, ce qui ne permet pas aux LEDs de bien correspondre à la couleur réelle affichée sur l'écran. Nous nous sommes intéressés à la méthode de balance des blancs, puis avons plutôt mis en place un recalibrage de couleur à l'aide de matrices de changement de base. À partir de la matrice de la couleur réelle (rouge, verte ou bleue), nous avons regardé la matrice de la couleur perçue par la caméra et avons déterminé la matrice permettant de passer de l'une à l'autre. La dimension ne correspondait pas et nous avons mis du temps à comprendre comment bien implémenter le code afin qu'il soit compatible avec la fonction d'acquisition d'image. La principale source d'erreur de cette méthode est due au fait qu'elle considère la conversion entre les matrices comme linéaire, ce qui n'est pas forcément le cas.

Enfin, le rafraîchissement ne fonctionnait pas au premier abord. En imposant une fréquence de rafraîchissement trop rapide, le programme ne suivait pas. Finalement, nous avons utilisé une boucle while, dont on contrôle la fréquence de parcours en utilisant la fonction time.sleep qui retarde la boucle de la durée souhaitée. Lorsque ce retard n'est pas assez important, le système ne suit pas. Il a fallu trouver un compromis entre un retard satisfaisant pour que le programme fonctionne et une

durée petite pour que les couleurs des LEDs correspondent le plus possible à celles affichées sur l'écran, lesquelles bougent lors du visionnage d'un film, par exemple.

7.2 Analyse du travail d'équipe

Chacun s'est attelé à une partie du code. La répartition a été faite souvent en binômes : ainsi, lors des premières séances, Taha et Alban ont codé l'acquisition de l'image et de la commande, d'une part, et Nathan et Flore se sont occupés du traitement de l'image et notamment du moyennage, d'autre part. Par la suite, les problèmes rencontrés ont précisé de nouvelles tâches : le problème de calibrage a été traité par Nathan et Flore, celui du rafraîchissement par Taha et Alban. Néanmoins, une discussion à quatre a toujours été tenue, ce qui a permis à chacun de contribuer à la résolution de chaque problème, et ainsi d'être au courant de l'avancée de chaque étape.

8. Conclusion

Ainsi, l'éclairage Ambilight prolonge l'image affichée sur l'écran à l'aide d'un bandeau de LEDs. Le projet repose sur trois étapes : l'acquisition d'image, le traitement de celle-ci et le pilotage des LEDs. Le programme est automatiquement rafraîchi afin que l'éclairage suive en temps réel les images affichées sur l'écran, notamment lors du visionnage de vidéos.
Ce projet nous a fait découvrir plusieurs dispositifs, comme la carte RaspberryPi, le système Linux ou encore les bandeaux de LEDs qui nécessitent un câblage particulier. En outre, les difficultés liées à la calibration et au rafraîchissement ont finalement été surmontées.
Cependant, des pistes d'amélioration sont possibles. Il serait notamment intéressant d'améliorer le découpage de l'écran, afin que seuls les bords soient pris en compte et donc prolongés. D'autre part, le calibrage des couleurs pourrait être amélioré via une autre méthode, afin que le bleu, par exemple, soit plus fidèle. Le rafraîchissement, quant à lui, serait difficile à améliorer, étant principalement limité par le matériel dont nous disposons.

Annexe:

Code python utilisé:

```
import cv2 as cv
import numpy as np
import math
import statistics as stat
import time
import smbus
from datetime import datetime
from rpi_ws281x import *

#Traitement de l'image
def traitementimage(Image):
    (ligne,colonne,pxl)=np.shape(Image)



    mi_l = ligne//2
    mi_c = colonne//2
    fin_l = round(ligne*0.05)  #round permet de prendre l'arrondi (1/20e de la
ligne à adapter éventuellement)
    fin_c = round(colonne*0.05)
    HG = []
    HD = []
    BG = []
    BD = []
    Tab_moy = []
#coté haut gauche
    for p in range (3):  #pour traiter chaque couleur une par une
        cote1 = np.mean(Image[0:fin_l,0:mi_c,p])
        cote2 = np.mean(Image[fin_l:mi_l,0:fin_c,p])
        HG.append(int(round(np.mean([cote1,cote2]))))  ## M est peut être un
flottant (le convertir en entier pour le PWM)
    Tab_moy.append(HG)
#coté haut droit
    for p in range (3):  #pour traiter chaque couleur une par une
        cote1=np.mean(Image[0:fin_l,mi_c:colonne,p])
        cote2= np.mean(Image[fin_l:mi_l,(colonne-fin_c):colonne,p])
        HD.append(int(round(np.mean([cote1,cote2]))))
    Tab_moy.append(HD)
#coté bas gauche
```

```
    for p in range (3):  #pour traiter chaque couleur une par une
        cote1=np.mean(Image[mi_l:ligne,0:mi_c,p])
        cote2= np.mean(Image[ligne-fin_l:ligne,fin_c: mi_c,p])
        BG.append(int(round(np.mean([cote1,cote2]))))
    Tab_moy.append(BG)
#coté bas droit
    for p in range (3):  #pour traiter chaque couleur une par une
        cote1=np.mean(Image[mi_l:ligne,colonne-fin_c:colonne,p])
        cote2= np.mean(Image[fin_l:ligne,mi_c:colonne-fin_c,p])
        BD.append(int(round(np.mean([cote1,cote2]))))
    Tab_moy.append(BD)

    return Tab_moy



def changement_couleur(liste): #liste=liste de 110 listes de 3 éléments
correspondant aux couleurs v,r,b
       for i in range (110):
        strip.setPixelColor(i, Color(liste[i][1], liste[i][2], liste[i][0]))
    strip.show()

def correction_couleur(image,mcc):
    image_c=image.reshape((image.shape[0]*image.shape[1],3))
    image_corrigee=np.matmul(image_c,mcc)
    return image_corrigee.reshape(image.shape).astype(image.dtype)

matrice_couleur=[[ 1.08817775e+00,  9.80373966e-04, -4.22961340e-03],
   [-6.08224008e-01,  1.48027505e+00, -5.57758091e-01],
       [ 1.25814659e-01, -3.06203469e-01,  1.32104925e+00]]

def recuperationphoto():

            cap=cv.VideoCapture(0)  #lancement de la caméra

            return_value,image = cap.read()     #capture une image de la
caméra


            return  image


while(1):
                        #configuration de la banderole LED
```

```
        LED_COUNT      = 110     # Number of LED pixels.
        LED_PIN        = 18      # GPIO pin connected to the pixels (must
support PWM!).
        LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually
800khz)
        LED_DMA        = 10      # DMA channel to use for generating signal
(try 10)
        LED_BRIGHTNESS = 255     # Set to 0 for darkest and 255 for brightes
        LED_INVERT     = False   # True to invert the signal (when using NPN
transistor level shift)
        LED_CHANNEL    = 0
        LED_STRIP      = ws.SK6812_STRIP_RGBW
            #LED_STRIP      = ws.SK6812W_STRIP

        if __name__ == '__main__':
                # crée un objet NeoPixel avec la bonne configuration
          strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA,
LED_INVERT, LED_BRIGHTNESS, LED_CHANNEL, LED_STRIP)
                # initialise la bibliothèque
            strip.begin()

Tab_moy=traitementimage(correction_couleur(recuperationphoto(),matrice_couleur
))
        for i in range (4):
            Tab_moy[i]=list(map(int,Tab_moy[i]))
        print(Tab_moy)
        liste_couleur=11*[Tab_moy[2]]+27*[Tab_moy[0]]+28*[Tab_moy[1]]
+27*[Tab_moy[3]]+17*[Tab_moy[2]]

        changement_couleur(liste_couleur)
        time.sleep(0.05) #retarde le parcours suivant de la boucle while.
```