Rapport Technique Vision Industrielle

SALY Nadira, YOULOU Rebecca, PEGON Etienne, AHMANE Ihssene

31 Mars 2023

Contents

1 Introduction

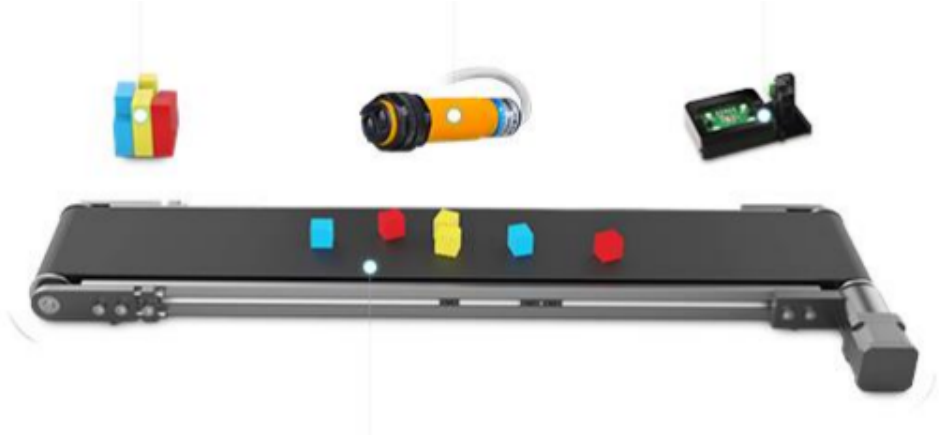

Figure 1: Les éléments à notre disposition pour réaliser le projet de Vision Industrielle
Source: lense.institutoptique.fr

1.1 Descriptif du projet

Ce rapport technique va porter sur le travail que le groupe G1.01 a effectué autour du projet Vision Industrielle. L'objectif du projet était d'automatiser une chaîne de tri, i.e. de pouvoir trier un nombre d'objets selon un critère qui pouvai être soit la couleur, soit la forme, soit les deux. Pour se faire, nous avions à notre disposition un **convoyeur** muni de **3 poussoirs**, une **caméra**, un **capteur de couleur**, un **détecteur de position** ainsi qu'une **carte Nucléo**.

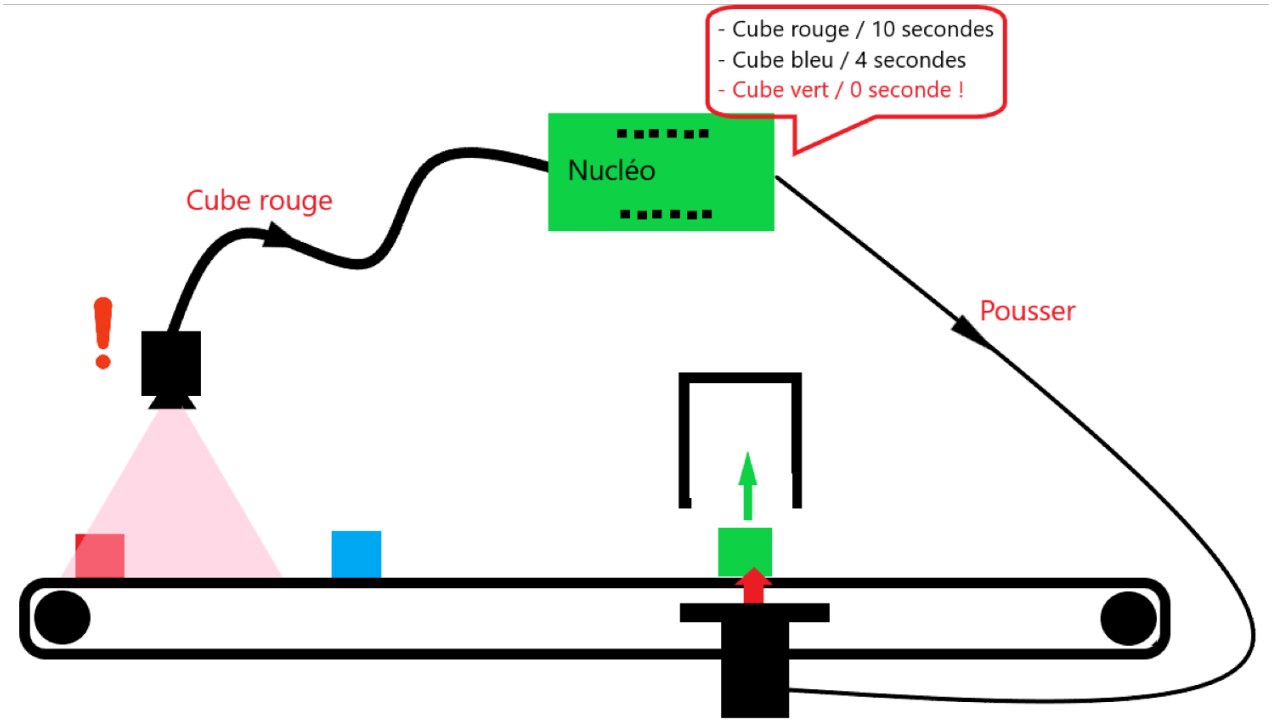

Figure 2: Schéma de principe de notre système

1.2 Notice d'utilisation du prototype

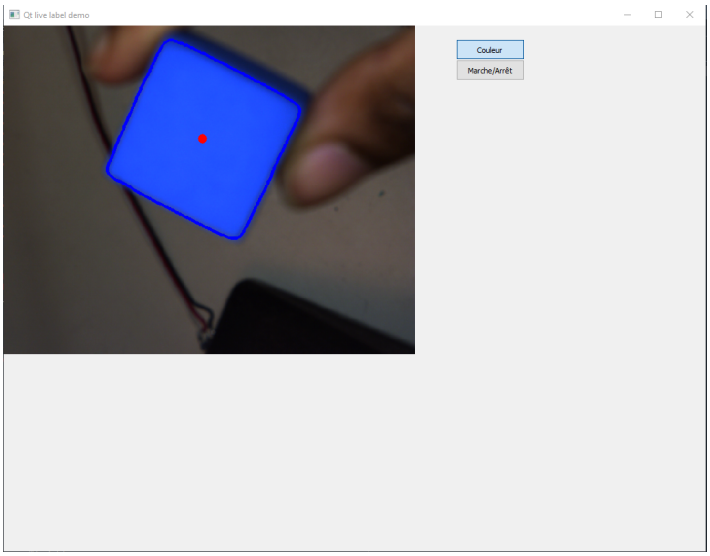

Figure 3: IHM simple que nous avons réalisée, avec cube détecté et centre calculé

Pour pouvoir utiliser notre prototype, il faut tout d'abord lancer le programme Python pour lancer l'IHM et la communication Ordinateur-Nucléo. Puis, il faut lancer le code compilé global contenant toutes les fonctions MBED compilées sur la Nucléo. Ce programme met en marche le tapis et écoute les informations provenant de l'ordinateur. En activant le bouton **détection de couleurs**, la caméra va relever le ou les couleurs présentes dans l'image et les envoyer à la carte nucléo et c'est comme cela que l'on va pouvoir détecter les couleurs.

1.3 Cahier des charges à respecter pour ce prototype

Le prototype que nous allons réalisé doit respecter plusieurs critères. Niveau **rapidité**, il doit être capable de trier 10 pièces par minute donc une pièce toutes les 6 secondes. Le système doit ensuite être capable de **détecter** et différencier les 4 couleurs de base que sont le rouge, le vert, le jaune et le bleu. On tolère ensuite une erreur de 1 pièce sur 1000 en matière de tri, et le développement d'une **Interface Homme-Machine** si possible, qui pourra faciliter l'utilisation du prototype par une personne qui ne l'a pas du tout développé et ne sais pas vraiment comment il fonctionne.

Le schéma fonctionnel associé à notre projet est alors celui décrit à la figure 4.

Il faudra noter que ce schéma fonctionnel représente ce que nous aurions développé comme prototype si nous avions le temps. Le prototype que nous vous proposons aujourd'hui est un version moins avancée que celle-là. Nous n'avons en effet pas eu le temps de développer l'IHM comme prévu et avons même rencontrés beaucoup de contre-temps que nous expliciterons plus dans le bilan. Nous étions tout de même pas si loin du but, mais les circonstances de cette année-là ont fait qu'il était compliqué pour notre groupe de développer tous les algorithmes nécessaires dans le temps imparti. (voir section3.1).

2 Description des algorithmes qui nous ont permis de réaliser notre prototype

Voici les algorithmes différents algorithmes que nous avons utilisé pour réaliser notre projet, tous découpés en grandes fonctionnalités.

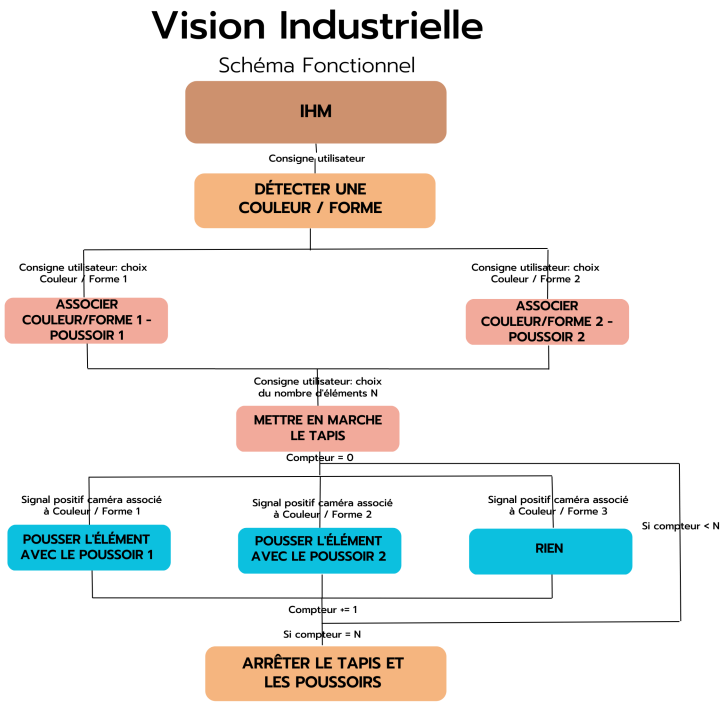

Figure 4: Schéma Fonctionnel idéal du prototype

2.1 Détection de couleurs sur Python

Il a tout d'abord fallu apprendre à récupérer le flux vidéo de la caméra uEye à l'aide du programme tutoriel de la caméra. Ce fut assez simple: il suffisait de conserver ce programme de base et de faire les opérations de traitement d'image expliquées ci-dessous dans la boucle while du programme de base (avant le cv2.imshow car cette commande consiste à afficher l'image que l'on met en argument donc, si l'on veut faire apparaître des modifications dessus, il faut les faire avant). On obtient le programme total suivant:

```
#Libraries
from pyueye import ueye
import numpy as np
import cv2
import sys

#-----------------------------------------------------------------------------

#Variables
hCam = ueye.HIDS(0)                    #0: first available camera;  1-254: The camera wi
sInfo = ueye.SENSORINFO()
cInfo = ueye.CAMINFO()
pcImageMemory = ueye.c_mem_p()
MemID = ueye.int()
```

```
rectAOI = ueye.IS_RECT()
pitch = ueye.INT()
nBitsPerPixel = ueye.INT(24)    #24: bits per pixel for color mode; take 8 bits p
channels = 3                    #3: channels for color mode(RGB); take 1 channel
m_nColorMode = ueye.INT()               # Y8/RGB16/RGB24/REG32
bytes_per_pixel = int(nBitsPerPixel / 8)
#---------------------------------------------------------------------------
print("START")
print()


# Starts the driver and establishes the connection to the camera
nRet = ueye.is_InitCamera(hCam, None)
if nRet != ueye.IS_SUCCESS:
    print("is_InitCamera ERROR")

# Reads out the data hard-coded in the non-volatile camera memory and writes it t
nRet = ueye.is_GetCameraInfo(hCam, cInfo)
if nRet != ueye.IS_SUCCESS:
    print("is_GetCameraInfo ERROR")

# You can query additional information about the sensor type used in the camera
nRet = ueye.is_GetSensorInfo(hCam, sInfo)
if nRet != ueye.IS_SUCCESS:
    print("is_GetSensorInfo ERROR")

nRet = ueye.is_ResetToDefault( hCam)
if nRet != ueye.IS_SUCCESS:
    print("is_ResetToDefault ERROR")

# Set display mode to DIB
nRet = ueye.is_SetDisplayMode(hCam, ueye.IS_SET_DM_DIB)

# Set the right color mode
if int.from_bytes(sInfo.nColorMode.value, byteorder='big') == ueye.IS_COLORMODE_BA
    # setup the color depth to the current windows setting
    ueye.is_GetColorDepth(hCam, nBitsPerPixel, m_nColorMode)
    bytes_per_pixel = int(nBitsPerPixel / 8)
    print("IS_COLORMODE_BAYER: ", )
    print("\tm_nColorMode: \t\t", m_nColorMode)
    print("\tnBitsPerPixel: \t\t", nBitsPerPixel)
    print("\tbytes_per_pixel: \t\t", bytes_per_pixel)
    print()

elif int.from_bytes(sInfo.nColorMode.value, byteorder='big') == ueye.IS_COLORMODE
    # for color camera models use RGB32 mode
    m_nColorMode = ueye.IS_CM_BGRA8_PACKED
    nBitsPerPixel = ueye.INT(32)
    bytes_per_pixel = int(nBitsPerPixel / 8)
```

```
        print("IS_COLORMODE_CBYCRY: ", )
        print("\tm_nColorMode: \t\t", m_nColorMode)
        print("\tnBitsPerPixel: \t\t", nBitsPerPixel)
        print("\tbytes_per_pixel: \t\t", bytes_per_pixel)
        print()

    elif int.from_bytes(sInfo.nColorMode.value, byteorder='big') == ueye.IS_COLORMODE_M
        # for color camera models use RGB32 mode
        m_nColorMode = ueye.IS_CM_MONO8
        nBitsPerPixel = ueye.INT(8)
        bytes_per_pixel = int(nBitsPerPixel / 8)
        print("IS_COLORMODE_MONOCHROME: ", )
        print("\tm_nColorMode: \t\t", m_nColorMode)
        print("\tnBitsPerPixel: \t\t", nBitsPerPixel)
        print("\tbytes_per_pixel: \t\t", bytes_per_pixel)
        print()

    else:
        # for monochrome camera models use Y8 mode
        m_nColorMode = ueye.IS_CM_MONO8
        nBitsPerPixel = ueye.INT(8)
        bytes_per_pixel = int(nBitsPerPixel / 8)
        print("else")

    # Can be used to set the size and position of an "area of interest"(AOI) within a
    nRet = ueye.is_AOI(hCam, ueye.IS_AOI_IMAGE_GET_AOI, rectAOI, ueye.sizeof(rectAOI)
    if nRet != ueye.IS_SUCCESS:
        print("is_AOI ERROR")

    width = rectAOI.s32Width
    height = rectAOI.s32Height

    # Prints out some information about the camera and the sensor
    print("Camera model:\t\t", sInfo.strSensorName.decode('utf-8'))
    print("Camera serial no.:\t", cInfo.SerNo.decode('utf-8'))
    print("Maximum image width:\t", width)
    print("Maximum image height:\t", height)
    print()

    #---------------------------------------------------------------------------

    # Allocates an image memory for an image having its dimensions defined by width a
    nRet = ueye.is_AllocImageMem(hCam, width, height, nBitsPerPixel, pcImageMemory, M
    if nRet != ueye.IS_SUCCESS:
        print("is_AllocImageMem ERROR")
    else:
        # Makes the specified image memory the active memory
        nRet = ueye.is_SetImageMem(hCam, pcImageMemory, MemID)
        if nRet != ueye.IS_SUCCESS:
```

```
        print("is_SetImageMem_ERROR")
    else:
        # Set the desired color mode
        nRet = ueye.is_SetColorMode(hCam, m_nColorMode)



# Activates the camera's live video mode (free run mode)
nRet = ueye.is_CaptureVideo(hCam, ueye.IS_DONT_WAIT)
if nRet != ueye.IS_SUCCESS:
    print("is_CaptureVideo_ERROR")

# Enables the queue mode for existing image memory sequences
nRet = ueye.is_InquireImageMem(hCam, pcImageMemory, MemID, width, height, nBitsPe
if nRet != ueye.IS_SUCCESS:
    print("is_InquireImageMem_ERROR")
else:
    print("Press_q_to_leave_the_programm")

#---------------------------------------------------------------------------

# Continuous image display
while(nRet == ueye.IS_SUCCESS):

    # In order to display the image in an OpenCV window we need to...
    # ...extract the data of our image memory
    array = ueye.get_data(pcImageMemory, width, height, nBitsPerPixel, pitch, cop

    # bytes_per_pixel = int(nBitsPerPixel / 8)

    # ...reshape it in an numpy array...
    frame = np.reshape(array,(height.value, width.value, bytes_per_pixel))

    # ...resize t                              he image by a half
    frame = cv2.resize(frame,(0,0),fx=0.5, fy=0.5)

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

#---------------------------------------------------------------------------
    #Couleur rouge

    lower_bound_red1 = np.array([175, 150, 150])
    upper_bound_red1 = np.array([180, 255, 255])
    lower_bound_red2 = np.array([0, 150, 150])
    upper_bound_red2 = np.array([5, 255, 255])
    mask_red1 = cv2.inRange(hsv, lower_bound_red1, upper_bound_red1)
    mask_red2 = cv2.inRange(hsv, lower_bound_red2, upper_bound_red2)
    mask_red = mask_red1 + mask_red2
```

```
    #Ce sont de pures op rations de traitement d'image qui permettent d'avoir de
    kernel = np.ones((7,7),np.uint8)

    mask_red = cv2.morphologyEx(mask_red, cv2.MORPH_CLOSE, kernel)
    mask_red = cv2.morphologyEx(mask_red, cv2.MORPH_OPEN, kernel)

    # Image contenant que les objets rouges (le reste est noir)
    segmented_img = cv2.bitwise_and(frame, frame, mask=mask_red)

    # Trouve le contour des objets d tect s
    contours_red, hierarchy_red = cv2.findContours(mask_red.copy(), cv2.RETR_EXTE

    # Permet d'avoir les coordonn es des objets rouges de l'image et de tracer l
    cv2.drawContours(frame, contours_red, -1, (0, 0, 255), 3) # Premier argument:
    # Deuxi me argument: contours que l'on souhaite tracer
    # Troisi me argument: couleur utilis e pour tracer les contours
    # Quatri me argument:   paisseur   des contours

#-------------------------------------------------------------------------------
    #Couleur jaune

    lower_bound_yellow = np.array([25, 150, 150])
    upper_bound_yellow = np.array([35, 255, 255])
    mask_yellow = cv2.inRange(hsv, lower_bound_yellow, upper_bound_yellow)

    # Ce sont de pures op rations de traitement d'image qui permettent d'avoir d
    mask_yellow = cv2.morphologyEx(mask_yellow, cv2.MORPH_CLOSE, kernel)
    mask_yellow = cv2.morphologyEx(mask_yellow, cv2.MORPH_OPEN, kernel)

    # Image contenant que les objets jaunes (le reste est noir)
    segmented_img = cv2.bitwise_and(frame, frame, mask=mask_yellow)

    # Trouve le contour des objets jaunes d tect s
    contours_yellow, hierarchy_yellow = cv2.findContours(mask_yellow.copy(), cv2.

    # Permet d'avoir les coordonn es des objets jaunes de l'image et de tracer l
    cv2.drawContours(frame, contours_yellow, -1, (0, 255, 255), 3)

#-------------------------------------------------------------------------------
    #Couleur verte

    lower_bound_green = np.array([50, 150, 150])
    upper_bound_green = np.array([60, 255, 255])
    mask_green = cv2.inRange(hsv, lower_bound_green, upper_bound_green)

    #Ce sont de pures op rations de traitement d'image qui permettent d'avoir de
    mask_green = cv2.morphologyEx(mask_green, cv2.MORPH_CLOSE, kernel)
    mask_green = cv2.morphologyEx(mask_green, cv2.MORPH_OPEN, kernel)
```

```
    # Image contenant que les objets verts (le reste est noir)
    segmented_img = cv2.bitwise_and(frame, frame, mask=mask_green)

    # Trouve le contour des objets verts d tect s
    contours_green, hierarchy_green = cv2.findContours(mask_green.copy(), cv2.RET

    # Permet d'avoir les coordonn es des objets verts de l'image et de tracer le
    cv2.drawContours(frame, contours_green, -1, (0, 255, 0), 3)
#-------------------------------------------------------------------------------
     #Couleur bleue

    lower_bound_blue = np.array([110, 150, 150])
    upper_bound_blue = np.array([120, 255, 255])
    mask_blue = cv2.inRange(hsv, lower_bound_blue, upper_bound_blue)

    # Ce sont de pures op rations de traitement d'image qui permettent d'avoir d

    mask_blue = cv2.morphologyEx(mask_blue, cv2.MORPH_CLOSE, kernel)
    mask_blue = cv2.morphologyEx(mask_blue, cv2.MORPH_OPEN, kernel)

    # Image contenant que les objets bleus (le reste est noir)
    segmented_img = cv2.bitwise_and(frame, frame, mask=mask_blue)

    # Trouve le contour des objets bleus d tect s
    contours_blue, hierarchy_blue = cv2.findContours(mask_blue.copy(), cv2.RETR_EX

    # Permet d'avoir les coordonn es des objets bleus de l'image et de tracer le
    cv2.drawContours(frame, contours_blue, -1, (255, 0, 0), 3)


#-------------------------------------------------------------------------------
    # M = cv2.moments(contours_red)
    # if M["m00"] != 0:
    #     cX_red = int(M["m10"] / M["m00"])
    #     cY_red = int(M["m01"] / M["m00"])

    #     cv2.putText(frame, "Rouge", cX_red, cY_red, cv2.FONT_HERSHEY_SIMPLEX, 0.

#-------------------------------------------------------------------------------
    # Affiche la vid o avec toutes les modifications faites au pr alable
    cv2.imshow("SimpleLive_Python_uEye_OpenCV", frame)

    # Press q if you want to end the loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

#-------------------------------------------------------------------------------

# Releases an image memory that was allocated using is_AllocImageMem() and remove
```

```
ueye.is_FreeImageMem(hCam, pcImageMemory, MemID)

# Disables the hCam camera handle and releases the data structures and memory are
ueye.is_ExitCamera(hCam)

# Destroys the OpenCv windows
cv2.destroyAllWindows()

print()
print("END")
```

Ci-dessous, on expliquera les points qui ont permis de traiter les images.

Pour pouvoir détecter les couleurs des objets que nous avons à trier, nous avons décidé d'utiliser un algortihme de détection de couleurs basé sur Python. Cet algorithme prend les images issues de la vidéo, les convertit en HSV et effectue un masquage en fonction de si oui ou non la couleur correspond au valeurs HSV définies au préalable.

Le format HSV (Hue Saturation Value) est le format le plus adéquat pour détecter une couleur sur une image. En effet, il est composé de 3 paramètres différents: la **teinte**, le paramètre qui va permettre de différencier les couleurs: rouge, jaune, bleu, vert, orange, etc; la **saturation**, qui va permettre de définir l'intensité de la couleur, variant de 0 à 100%; la **valeur**, qui définit la brillance de la couleur. Le code ci-dessous est le code générique utilisé pour identifier les 4 couleurs:

```
#Libraries
from pyueye import ueye
import numpy as np
import cv2
import sys

#Couleur rouge

lower_bound_red1 = np.array([175, 150, 150])
upper_bound_red1 = np.array([180, 255, 255])
lower_bound_red2 = np.array([0, 150, 150])
upper_bound_red2 = np.array([5, 255, 255])
mask_red1 = cv2.inRange(hsv, lower_bound_red1, upper_bound_red1)
mask_red2 = cv2.inRange(hsv, lower_bound_red2, upper_bound_red2)
mask_red = mask_red1 + mask_red2


#Ce sont de pures op rations de traitement d'image qui permettent d'avoir des co
kernel = np.ones((7,7),np.uint8)

mask_red = cv2.morphologyEx(mask_red, cv2.MORPH_CLOSE, kernel)
mask_red = cv2.morphologyEx(mask_red, cv2.MORPH_OPEN, kernel)
```

```
# Image contenant que les objets rouges (le reste est noir)
segmented_img = cv2.bitwise_and(frame, frame, mask=mask_red)

# Trouve le contour des objets d tect s
contours_red, hierarchy_red = cv2.findContours(mask_red.copy(), cv2.RETR_EXTERNAL

# Permet d'avoir les coordonn es des objets rouges de l'image et de tracer les c
cv2.drawContours(frame, contours_red, -1, (0, 0, 255), 3) # Premier argument: ima
# Deuxi me argument: contours que l'on souhaite tracer
# Troisi me argument: couleur utilis e pour tracer les contours
# Quatri me argument:    paisseur    des contours
```

Les tableaux définies dans les premières lignes du code contiennent les valeurs respectivent de H, de S et de V dans cet ordre, codé de 0 à 180° pour H (PAS 360° dans Python) et de 0 à 255 pour S et V. Les valeurs ont été ajustées en fonction de l'éclairage ambiant de façon à détecter la bonne valeur de rouge (resp. vert, jaune, bleu) et uniquement la bonne valeur de rouge (resp. vert, jaune, bleu). La particularité pour la couleur rouge c'est qu'elle se trouve sur le bord du cercle (entre [0°,15°] et [165°, 180°]) d'où la nécessité de créer deux masques et les assembler en les sommant pour pouvoir détecter la couleur demandée. Pour les autres, un seul masque est créé.

La ligne segmented_img est l'image obtenue après application du masque de couleur sur l'image de base: les objets en blanc sur l'image sont les objets rouges, et le reste est noir car pas de la bonne couleur. cv2.findContours permet ensuite de récuperer les contours des objets rouges (qui ont satisfait la condition HSV précédente) et cv2.drawContours permet de tracer ces contours pour l'interface homme-machine:les objets détectés apparaissent entourés pour que l'utilisateur comprenne qu'ils ont bien été vus et leur couleur reconnue(cf. figure 5).

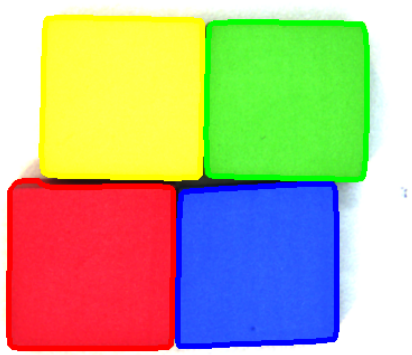

Figure 5: Les 4 cubes de couleurs primaires détectés

```
for ind in range(4) :                                    # Pour chaque
```

```
contours = [contours_red, contours_yellow, contours_green, contours_blue][ind
#print("boucle for")
for i in contours:                                                    # Pour chaque
    M = cv2.moments(i)                                                # On trouve l
    if M['m00'] != 0:
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        cv2.circle(frame, (cx, cy), 7, (0, 0, 255), -1)
        couleur_du_centre.append(ind)                                 # Et dans une
```

Dans la volée, nous avons décidé d'ajouter une partie de code qui permettrait de calculer le centre des cubes présents sur l'image. Pourquoi ? Parce que c'est le système que nous avons décidé d'utiliser pour repérer les cubes sur le tapis et calculer le temps d'attente nécessaire pour que le cube soit poussé par le bon poussoir.

On va mettre tous les contours déterminés dans l'image dans le tableau "contours". Une fois ce tableau rempli, on va le parcourir un à un et calculer les coordonnées du baricentre du contour correspondant et on trace un point au centre du cube étudié. Puisqu'on enregistre toujours les contours dans le même ordre (comme montré dans la ligne "contours = [contours_red, contours_yellow...]", si le moment M['m00'] n'est pas nul pour le premier élément de contours (condition du if vraie), alors on a du rouge, si c'est pour le 2e élément, alors on a du jaune, etc. Cela permet de connaître exactement la couleur du ou des cube.s présent.s sur l'image.

2.2 Interface Homme-Machine sur Python

L'interface Homme-Machine est une interface qui permet de rendre l'utilisation d'un logiciel plus simple pour un utilisateur lambda, qui n'aurait pas à modifier des paramètres dans le code par exemple. Notre IHM permet d'afficher la vidéo filmée par la caméra et de sélectionner le mode de détection. Une fois la détection de couleur activée, les objets de la couleur que l'on souhaite trier sont entourés de la couleur qu'ils sont pour montrer simplement à l'utilisateur qu'ils ont bien été détectés.

2.2.1 Les bibliothèques importées

```
## Biblioth ques import es

# On a utilis  Qt5 pour faire notre interface et en particulier la bibloth que
from PyQt5 import QtGui
from PyQt5.QtWidgets import QMainWindow, QWidget, QApplication, QLabel, QVBoxLayo
from PyQt5.QtGui import QPixmap, QIcon
from PyQt5.QtCore import pyqtSignal, pyqtSlot, Qt, QThread

import sys

# On a utilis  cv2 pour le traitement d'image
import cv2
```

```
import numpy as np
import time


# L'utilisation de la cam ra uEye n cessite une biblioth que sp ciale (pas le
from pyueye import ueye

# On a utilis  PySerial pour la communication entre le PC et la nucl o
from serial import Serial
import serial.tools.list_ports
```

On a utilisé Qt5 pour créer la fenêtre et tout ce qui est dessus car c'est ce qui fonctionne le mieux pour intégrer un flux vidéo. De plus, il y a beaucoup de ressources sur internet car c'est ce qui est le plus communément utilisé. On doit importer la bibliothèque de la caméra uEye car elle ne s'utilise pas aussi simplement qu'une webcam. C'est un point qui nous a posé de gros soucis. En effet, le démarrage et la récupération de l'image demandent une procédure assez complexe pour la uEye et on a donc décidé de créer un thread spécifique à la récupération des images de la caméra. Celui-ci intègre toute la procédure d'ouverture de la communication avec la caméra, de configuration d'un lieu de stockage et la boucle qui récupère les images (comme expliqué dans la section 2.1). De plus, nous avons du mettre un temps d'inactivité sur ce thread entre chaque image car il récupère l'image puis l'envoie directement sans tenir compte de si le traitement de la précédente image est terminé ou non. Cela peut donc saturer l'ordinateur et l'interface se met à afficher une image qui est trop ancienne et le retard s'accumule. On a utilisé numpy pour quelques calculs sur l'image (qu'on a donc fait rentrer dans un tableau numpy) et time pour faire dormir le thread vidéo. On utilise PySerial pour communiquer avec la Nucléo.

2.2.2 Variables globales

```
## Variables globales

# La liste code_couleur sert  associer un nombre (l'indice)  une couleur (la
code_couleur = ["rouge", "jaune", "vert", "bleu"]
# Pareil mais avec l lettre  envoyer  la Nucl o
code_couleur_serial = ["R", "J", "V", "B"]
# Cette variable va servir  compter le nombre de cubes entr s dans le champ de
nb_centres_avant = 0

global selectPort
```

Les variables globales utilisées sont les suivantes : code_couleur sert à associer un nombre (l'indice) à une couleur (représentée ici par une chaîne de caractère), code_couleur serial fait de même mais la couleur est représentée par une lettre, à envoyer à la Nucléo(cf. 2.4, nb_centre_avant enregsitre le nombre de cube à l'image (utile pour détecter l'arrivée d'un nouveau cube) et enfin selectPort qui permet de transmettre l'information du port COM à utiliser pour communiquer avec la Nucléo.

2.2.3 Classes

```
## Classes (code r cup r  par internet et modifi  en grande partie pour notre

# Classe qui sert    contenir le processus de r cup ration des images (communic
# C'est un thread distinct, donc il faut limiter la vitesse de r cup ration des
class VideoThread(QThread):

    def __init__(self):

    # acquisition des images puis envoi vers le traitement
    def run(self):

    def stop(self):


# Classe qui repr sente l'application, qui est confondu avec la fen tre graphiq
# Plusieurs  tapes  : d finition de la fen tre, lancement du thread vid o, ou
class App(QMainWindow):
    def __init__(self):
```

La classe VideoThread contient dans run toute la procédure pour allumer la communication avec la caméra uEye puis récupérer les images et les renvoyer vers l'App (classe QMainWindow). De plus, elle contient un booléen qui indique si elle est en fonctionnement ou non. Elle est appelée dans un thread, c'est-à-dire un processus qui fonctionne en parallèle (indépendamment) du programme principal. Aussi, il y a une procédure spéciale pour couper la communication. Le programme prend par défaut la caméra uEye listée en position 0 (première position) car il est assez improbable d'avoir plusieurs caméras uEye sur le PC. Cela peut se modifier directement dans le code ligne 52. La classe App est celle qui contient le programme principal dont la fenêtre fait partie. On commence par définir la fenêtre, à laquelle on ajoute des boutons qu'on relie à des fonctions (des méthodes en fait ici). On lance le VideoThread de récupération des images, qu'on connecte à la fonction de traitement d'image. Enfin, on liste les ports com afin que l'utilisateur puisse saisir celui de la Nucléo. On garde alors en mémoire pour toutes les autres fonctions le port dans la variable globale selectPort.

2.2.4 Fonctions

```
## Fonctions (m thodes    l'int rieur de la classe)

    # Fonction qui sert    arr ter le tapis
    # La fonction laisse penser qu'on peut lancer et arr ter le tapis mais en pr
    @pyqtSlot()
    def tapistoggle(self):

    # Fonction qui  teint  le thread vid o
    def closeEvent(self, event):

    # Mise    jour de l'image, cela inclut donc le traitement d'image puis l'envo
    @pyqtSlot(np.ndarray)
    def update_image(self, frame):
```

```
            [Detections des contours et leur couleur]
            [Detection des centres]

            #Detection de l'arriv e d'un nouveau cube
            global nb_centres_avant
            elements_entres = len(couleur_du_centre) - nb_centres_avant
            nb_centres_avant = len(couleur_du_centre)
# On calcule si il y a plus de cubes qu'   l'image d'avant
            print(elements_entres)
            if elements_entres > 0 :
# Si il y a un nouveau cube dans le champs de vision
                for i in range(elements_entres) :
# On doit l'annoncer    la Nucl o avec sa couleur
                    ind_couleur_centre = couleur_du_centre[-i]

        # Fonction de transtypage de l'image de cv vers Qt
        def convert_cv_qt(self, frame):
```

Ces fonctions sont en réalités des méthodes associées à la classe App.
tapistoggle : on ouvre la communication avec le port com et on envoie la lettre qui correspond à arrêter le tapis :"S". De plus, on lit la réponse de la Nucléo car si elle n'est pas lue, le programme est bloqué à cause de l'instruction serNuc.inWaiting dans la boucle while (cf. paragraphe **??**).
update image : on regarde si le bouton de traitement couleur est activé et si il l'est, on fait le traitement. Nous réalisons les opérations décrites dans le paragraphe 2.1. Ensuite, nous enregistrons les centres des cubes en les repérant par leur couleur. Cela nous donne une liste des couleurs des cubes passés devant la caméra, par ordre d'arrivée. Cela permet donc de détecter qu'un nouveau cube est arrivé et sa couleur. On envoie directement l'information à la Nucléo qui s'occupe de tout le reste (cf. paragraphe 2.4).
convert cv qt : on change le type de l'image afin de pouvoir l'afficher dans la fenêtre Qt5.

2.2.5 Code qui lance l'App

```
## Code de lancement de l'application
if __name__=="__main__":
    app = QApplication(sys.argv)
    a = App()
    a.show()
    sys.exit(app.exec_())
```

On notera que ce code intègre directement le code de détection de couleurs car on avait déjà fini de travailler dessus. On a en fait déjà réalisé une partie de l'intégration à ce stade.

2.3 Tapis roulant (ou convoyeur) et poussoirs sur MBED (en C++)

Pourquoi est-il nécessaire d'avoir un tapis roulant et de poussoirs dans ce projet ? Parce que notre but est d'automatiser une chaîne de tri et que, pour ce faire, nous avons besoin que les objets à trier puissent arriver seuls dans la boîte correspondante à leur couleur. Il est donc nécessaire de les faire avancer

de la caméra (qui nous permet d'identifier leur couleur) au poussoir (qui les éjectera dans la bonne boîte).

2.3.1 Branchements à respecter pour faire fonctionner le convoyeur

Le convoyeur fonctionne avec une première carte spécifique qui fera le lien avec le tapis et une seconde qui fera le lien avec la carte Arduino. Sur la première, il faut brancher une tension en entrée de 5V, puis connecter les 4 ports à ceux du convoyeur (comme indiqué sur la figure 6). Sur la seconde, il y a 9 ports : 5V, GND, Clock), Enable, Control, Reset, Half, CW et Vref. Enable, Control, Reset et Vref sont branchés sur 5V. Half est branchée sur 0V. CW peut être branchée sur l'un des deux, suivant le sens du tapis que l'on souhaite. GND est branché à la masse. Et enfin, clock est à brancher à une sortie de la carte Nucléo, sortie qui sera utilisée dans le programme MBED pour pouvoir contrôler le tapis.

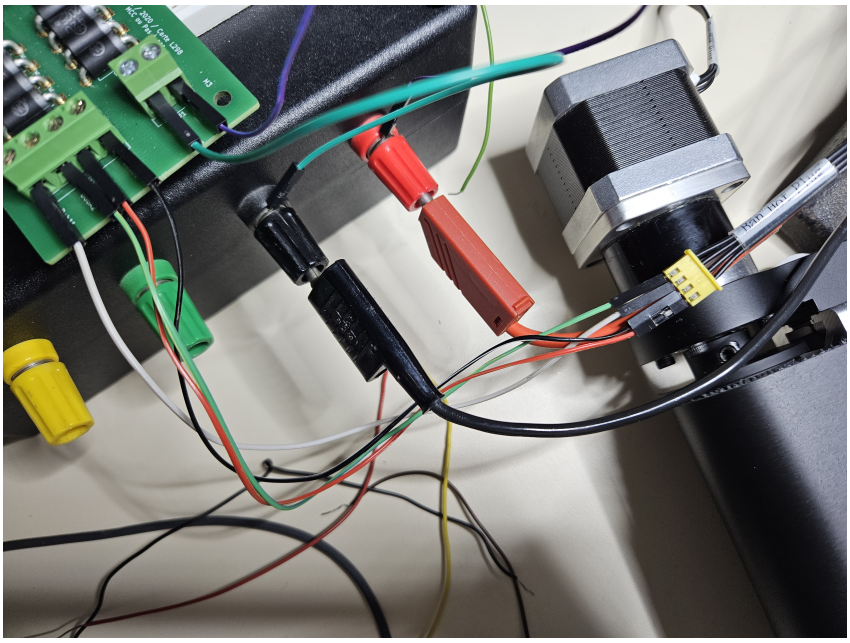

Figure 6: Branchements à suivre pour connecter la première carte spécifique avec le convoyeur

2.3.2 Branchements à respecter pour faire fonctionner un des poussoirs

Les poussoirs sont très facile à brancher. Ils fonctionnent également à l'aide d'un servomoteur. Ce dernier possède 3 fils: 2 fils d'alimentation (branchés sur 5V et sur Ground) et 1 fil de commande (en 0-5V) que l'on connecte à une sortie digitale (cf. figure 7).

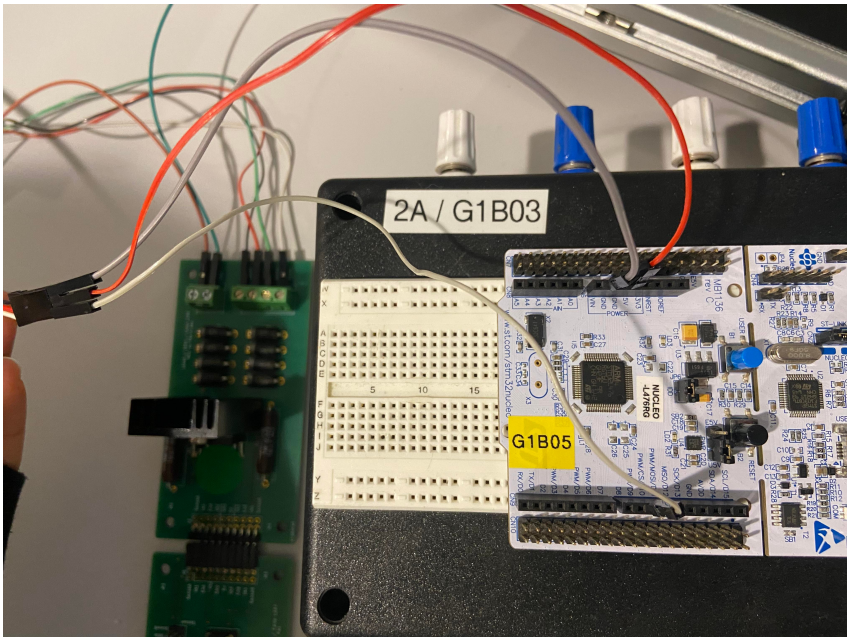

Figure 7: Branchement à suivre pour pouvoir contrôler les poussoirs

2.3.3 Tapis roulant sur MBED (en C++)

Mettre le tapis roulant en marche a été assez complexe pour nous d'une séance à l'autre. Cependant, le code reste assez simple et compréhensif.

```
## Code de mise en marche du tapis

#include "mbed.h"

// Pwm : modulation de largeurs d'impulsions

PwmOut servo_mot(D8);  // sortie du moteur sur la branche D8 de la carte Nucleo

int main(){
    servo_mot.period_us(3000);
    servo_mot.write(0.5);
    while(1){
    }
}
```

La première commande à l'intérieur du main permet de contrôler la période de rotation du moteur. En effet, afin de retrouver la vitesse du tapis à partir de cette fréquence, Le moteur pas à pas est constitué de 200 bobines placées toutes à $2 * \pi/200$. Pour une fréquence $f = 200$ hertz, on va lui donner 200 impulsions par secondes et il tournera d'un 1/2 tour donc de $\pi\, rad$. Etant donné que la liaison entre le moteur pas à pas et le tapis du convoyeur est une liaison pivot, la relation entre le déplacement angulaire du moteur et le déplacement du tapis est la suivante:

$$d = R_{moteur}.\alpha$$

avec d la distance dont s'est déplacée le tapis, R moteur le rayon du moteur et alpha l'angle dont a tourné le moteur pendant un temps t. D'après l'explication précédente, cet angle vaut:

$$\alpha = \frac{2.\pi.f.t}{N}$$

,avec N le nombre de bobines du moteur.

La seconde ligne permet de forcer le rapport cyclique à 0,5 (il correspond à la proportion d'état haut par rapport au total).

2.3.4 Poussoirs sur MBED (en C++)

Le code que l'on a utilisé afin de les faire fonctionner est le suivant :

```
## Code de mise en marche des poussoirs

#include "mbed.h"

// But : Faire fonctionner les poussoirs permettant de trier les cubes


//PwmOut servo_mot(D10);

int main() {
    int i, time = 1500;
    servo_mot.period_ms(20);          // Initialisation p riode
    servo_mot.pulsewidth_us(2200);    // Initialisation en position 0

    while(1){
        servo_mot.pulsewidth_us(700);    // Angle n gatif
        wait_us(1200000);
        servo_mot.pulsewidth_us(2200);    // Angle positif
        wait_us(500000);
        break;
    }
}
}
```

Afin de faire fonctionner le moteur, nous pilotons son angle de sortie compris entre 0° et 90°. La définition d'un angle positif/négatif va dépendre de la façon dont le poussoir a été conçu: si la tige du poussoir se trouve à gauche de sa roue, alors les angles sont inversés (on va de 90° à 0° au lieu d'aller de 0° à 90° en terme de code). A l'inverse, si la tige du poussoir se trouve à droite, les angles sont dans le "bon sens". Au départ, le poussoir se trouve à un certain angle, angle réellement codé au niveau du moteur (si l'on donne la commande au poussoir d'aller à l'angle repéré par "500" par exemple, il essaiera d'aller à la même position, peu importe sa position de départ). Ce n'est pas une différence d'angle que l'on demande en commande mais bien un angle précis que l'on souhaite atteindre. Pour que cet angle soit toujours le même et que l'on puisse garantir de la répétabilité de notre code, on initialise l'angle en lui donnant une première valeur de consigne avec la commande "servo_mot.pulsewidth_us(2200)". Ensuite, nous donnons l'ordre au moteur de tourner jusqu'à se retrouver à un angle de 700 (cet angle est indiqué comme négatif car 700 est inférieur à 2200 mais donner cette consigne faisait avancer notre tige parce que ce poussoir avait la particularité d'être monté à l'envers par rapport aux deux autres: un angle de consigne positif

faisait reculer la tige et un ordre de consigne négatif la faisait avancer). Nous attendons ensuite 1,2s avec la fonction wait car ce poussoir était très lent et c'était la durée nécessaire pour permettre à la tige d'atteindre la consigne angulaire comme demandée. Les deux autres lignes fonctionnent de la même façon et nous permettent de retourner à l'angle de départ (retirer le poussoir du tapis). Nous laissons cette fois-ci uniquement 0,5s pour réaliser l'instruction car le poussoir est beaucoup plus rapide au retour qu'à l'aller. A noter que l'association de la boucle while et de l'instruction break est totalement inutile dans cette partie et peut être enlevée sans soucis.
La difficulté de cette partie a été la nouvelle caractérisation de chaque poussoir à chaque séance car les valeurs variaient, probablement dû à la position initiale de la plaque rotative qui n'était pas la même et qui faussait les valeurs des angles. Un conseil que l'on peut vous donner est alors de vraiment faire attention à la position de départ de la tige du poussoir et d'essayer d'avoir la même à chaque fois (quitte à l'ajuster manuellement) car sinon, cela pourrait vous pousser à réaliser une caractérisation de chaque poussoir à chaque séance, ce qui est purement une perte de temps.

2.4 Phase d'intégration

Voici venir l'étape la plus délicate de notre projet: mettre en commun toutes les briques présentées ci-dessus pour obtenir notre prototype final. Le plus gros défi a alors été de trouver un moyen pour deux codes écrits dans des langages différents, que sont Python et C++, de communiquer. Pour se faire, nous avons dû apprivoiser le **protocole RS232** et les bibliothèques **Pyserial** et **UnbufferedSerial** associées respectivement à Python et à C++.

2.4.1 Code général Python

Nous avons décidé d'envoyer une lettre associée à chaque couleur de la caméra vers la carte nucléo contenue dans la variable data_to_send: 'R' pour rouge, 'V' pour vert, 'J' pour jaune et 'B' pour bleu. Cela se réalisait simplement grâce au code suivant:

```
global selectPort
serNuc = Serial('COM'+str(selectPort), 115200)  # Under Windows only
appOk = 1
while appOk:
    data_to_send = code_couleur_serial[ind_couleur_centre]
    if data_to_send == 'k' or data_to_send == 'K':
        appOk = 0
    else:
        serNuc.write(bytes(data_to_send, 'ascii'))
        while serNuc.inWaiting() == 0:
            pass
        data_rec = serNuc.read(1)  # bytes
        print(str(data_rec))
        appOk = 0
serNuc.close()
```

Cette séquence est ce qu'on utilise pour envoyer les données à la Nucléo. Cette communication suit le protocole RS232 et passe par le cable USB qui alimente la Nucléo. L'intégration de ce code n'a posé aucun problème puisqu'il est placé à la fin du traitement d'image et ouvre et ferme la communication donc il n'y a pas de risque de mauvaise terminaison du processus.

Nous avons réalisé très tôt l'intégration du traitement d'image dans le code de l'interface, ce qui a permis que ce soit assez simple. De plus nous avons presque tout mis dans la même fonction ce qui fait que le code de traitement d'image a pu être réutilisé tel quel, avec une simple fonction pour convertir au niveau des types, l'image dans le type nécessaire à l'algorithme de traitement.

2.4.2 Code général C++

Du côté de la carte Nucléo, il a fallu regrouper toutes les fonctions développées précédemment dans un même code. C'est ce que nous avons fait et nous obtenons le main suivant:

```
/* mbed Microcontroller Library
 * Copyright (c) 2019 ARM Limited
 * SPDX-License-Identifier: Apache-2.0
 */

//#include "mbed.h"
//#include "bibli.h"
#include "mbed.h"
#include "UnbufferedSerial.h"
#include <ctime>

PwmOut servo_mot_tapis(D8);
PwmOut servo_mot_poussoir_bleu(D10);
//PwmOut servo_mot_poussoir_jaune(D11);
PwmOut servo_mot_poussoir_rouge(D6);
char data;
char tab_couleur[10];
int tab_time[10];
int i(0);
    //clock_t sortie, pousse;
Timer t;
int date;


// Durees    calibrer en fonction du poussoir
int duree_rouge = 5000500; //1.75 secondes entre le temps ou le cube entre sous la camera et le
//int duree_jaune = 10000; //pareil version jaune
//int duree_vert = 15000; //sert a rien car ce cube tombe au bout du tapis
int duree_bleu = 3005000; //pareil version bleu
// On va appliquer cette fonction    chaque    lment    de la liste de cubes pass s devant la ca

int precision = 200000;


UnbufferedSerial py_port(USBTX, USBRX);

//Timer t;
//void ISR_my_pc_reception(void);
void test_poussoir(char couleur, int duree);
void poussoir_bleu(PwmOut servo_mot_poussoir1);
void poussoir_rouge(PwmOut servo_mot_poussoir3);
void poussoir_jaune(PwmOut servo_mot_poussoir2);
void ISR_py_port_reception(void);
```

```
int main()
{
    t.start();
    //while read

    py_port.baud(115200);
    py_port.attach(&ISR_py_port_reception, UnbufferedSerial::RxIrq);


    //servo_mot_poussoir_rouge.period_ms(20);          // Initialisation p riode

    servo_mot_tapis.period_us(2000);    // contr ler la p riode de rotation du moteur
    servo_mot_tapis.write(0.5);
    //servo_mot.write(0.5);    // forcer le rapport cyclique    0.5 (proportion d' tat  haut par


    //poussoir_bleu(servo_mot_poussoir_bleu);
    //poussoir_rouge(servo_mot_poussoir_rouge);

    while(1){



        date = t.elapsed_time().count();

        // date = t.read_us()

        for(int j(0); j < i; j++) {test_poussoir(tab_couleur[j], date - tab_time[i]);}
        //poussoir_rouge(servo_mot_poussoir_rouge);

        }

    }

}
```

Voici plus de détails sur ce que fait ce programme (via la fonction main): Ce programme initialise tout d'abord la communication avec l'ordinateur via le port défini à l'aide de la ligne "UnbufferedSerial py_port(USBTX, USBRX)". Ensuite, on met le tapis en marche comme décrit précédemment dans la section 2.3.3, puis on rentre dans la boucle while qui permet au tapis de continuer à rouler. Pendant que le tapis roule, on peut recevoir de nouvelles informations concernant les couleurs, comme décrit à cet endroit-là et on peut aussi déclencher un poussoir si jamais un cube doit être poussé.

2.4.3 Comment déclencher le bon poussoir ?

Pour déclencher le bon poussoir au bon moment, on a créé une fonction test_poussoir dont nous vous avons remis le code ci-dessous:

```
/* mbed Microcontroller Library
 * Copyright (c) 2019 ARM Limited
 * SPDX-License-Identifier: Apache-2.0
```

```
 */

//#include "mbed.h"
//#include "bibli.h"
#include "mbed.h"
#include "UnbufferedSerial.h"
#include <ctime>

PwmOut servo_mot_tapis(D8);
PwmOut servo_mot_poussoir_bleu(D10);
//PwmOut servo_mot_poussoir_jaune(D11);
PwmOut servo_mot_poussoir_rouge(D6);

void test_poussoir(char couleur, int duree){
    if (couleur == 'R')
    {
        wait_us(duree_rouge);
        poussoir_rouge(servo_mot_poussoir_rouge);
    } /* On attend que le cube rouge se trouve juste
    devant le poussoir avant de l'actionner hitoire de
    pousser au bon moment*/

    //if (couleur = "J" && abs(duree-duree_jaune)<precision) {/*lancer le poussoir jaune*/}

    if (couleur == 'B')
    {
        wait_us(duree_bleu); poussoir_bleu(servo_mot_poussoir_bleu);
    } /* On attend que le cube bleu se trouve juste devant
    le poussoir avant de l'actionner hitoire de pousser au
    bon moment*/
}
```

Une fois une donnée de couleur reçue par l'ordinateur, cette fonction va tester sa valeur pour reconnaître la couleur du cube si c'est un cube qui rentre, ou biens stopper le tapis (fonctionnalité pas encore implémentée). Une fois la couleur reconnue, elle va pouvoir déclencher le bon poussoir au bon moment grâce à l'instruction wait située dans toutes les boucles if
.

2.4.4 Réception des données

La partie nouvelle du code qui va permettre de récupérer les informations est la suivante:

```
void ISR_py_port_reception(void){
    py_port.read(&data, 1);
    /*if(data == 'S'){ //Stopper le tapis (normalement)
                break;
            }*/
    /*else*/ if(data == 'R' || data == 'J' || data == 'V' || data == 'B')
    {
        tab_couleur[i]=data; tab_time[i] = t.elapsed_time().count(); i++;
    }
    py_port.write(&data, 1);
```

```
}
```

Cette fonction permet d'initialiser la communication avec l'ordinateur. On lit la donnée lorsqu'elle arrive et on la renvoie à l'ordinateur avec l'instruction py_port.write(data, 1) à cause de l'instruction serNuc.inWaiting de Python contenue dans la boucle while. Si on ne le fait pas, on reste bloqué dans la boucle while car elle attend un retour de la part de la carte Nucléo après envoi d'une information.

3 Bilan

Avant de commencer à nous répartir les tâches, nous nous sommes demandés quels étaient nos objectifs/attentes concernant ce projet, dans quelle mesure nous considérerions que notre projet serait réussi. Il était aussi question de la réussite de notre prototype.

- Nous considérerons que notre **prototype** est réussi si l'on parvient à faire parfaitement tout ce qui est décrit dans le schéma fonctionnel ci-dessus. Il faudrait que l'IHM fonctionne exactement comme nous l'avions envisagé, c'est-à-dire que l'on puisse sélectionner le mode de fonctionnement (détection de couleurs ou détection de formes), les couleurs à trier, le nombre d'objets, que le tapis se mette en marche automatiquement, que les poussoirs trient correctement les objets selon leurs couleurs et que le tapis s'arrête directement une fois le nombre d'objets trié atteint.

- Nous considérerons que notre **projet** est réussi si l'on parvient à trouver une façon de s'organiser tou.te.s ensemble et à utiliser les briques que chacun a développé pour obtenir un produit qui fonctionne pas trop mal: le tapis qui avance et les cubes qui sont bien poussés correctement. On ne recherche pas du tout la même "perfection" que pendant le prototype. Dans cette catégorie, on vise surtout la capacité que l'on aurait eu à travailler en équipe même si le résultat n'est pas forcément à la hauteur de nos espérances décrites dans le prototype.

Au final, nous avons réussi notre projet, comme défini ci-dessus, mais pas tout à fait notre prototype. Nous n'avons pas pu développer suffisamment l'Interface Homme-Machine pour y parvenir, ni prévoir certaines difficultés (cf. 3.1).

La répartition initiale des tâches a ensuite été établie de la façon suivante:

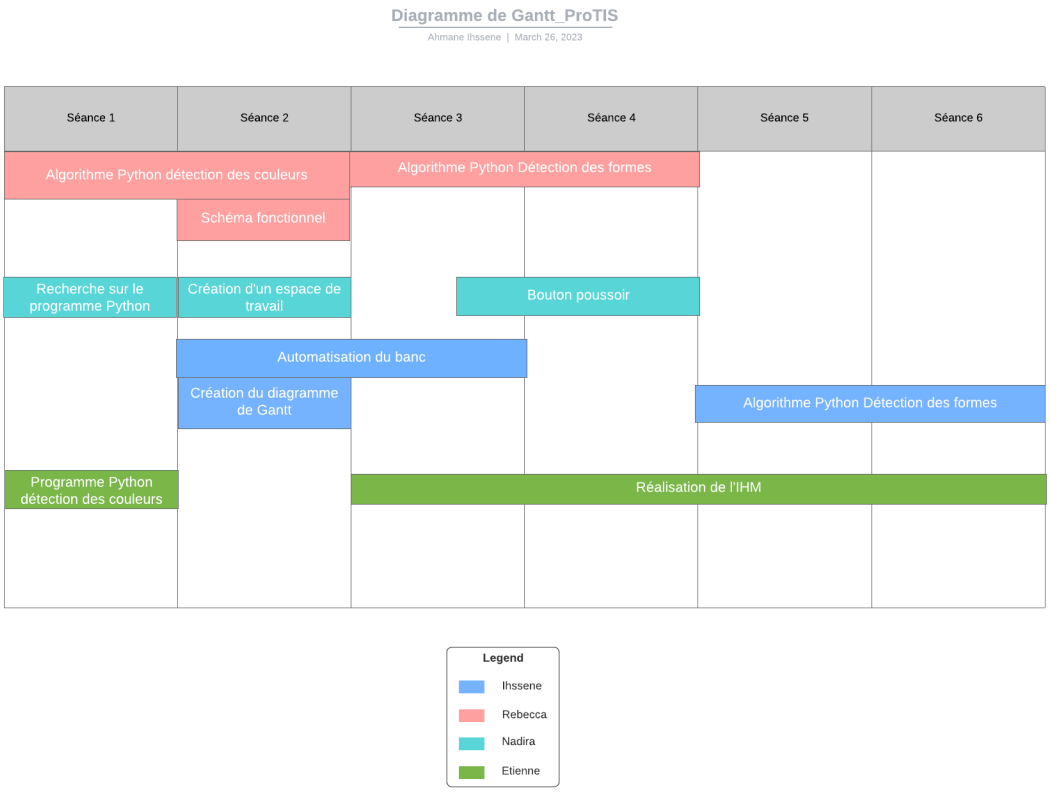

Figure 8: Répartition des tâches au sein de notre équipe

Nous sommes parvenus à respecter les deadlines que nous nous sommes imposées pour ces tâches (sauf celle pour l'intégration), i.e. nous avons tout fait en temps et en heure, mais la répartition des tâches n'a pas pu être respectée à cause de la grève, qui a fortement touché notre équipe puisque nous étions souvent deux personnes, voire même une seule personne de l'équipe sur 4 à être présente à la séance. Nous avons aussi pas spécifiquement alloué de temps à l'intégration mais nous le faisions en parallèle du développement de l'IHM.

3.1 Difficultés rencontrées

Nous avons rencontré pas mal de difficultés sur notre route vers la réussite de ce projet. La grève, que nous avons mentionné ci-dessus, nous a freiné dans notre élan et a rendu la tâche plus ardue. Voici les autres difficultés auxquelles nous avons été confronté et que nous sommes, ou non, parvenus à régler durant ces 6 semaines de projet.

On notera par exemple que nous avons été obligés d'utiliser la bibliothèque uEye pour pouvoir faire fonctionner la caméra de cette marque. Cependant, son utilisation est très différente de celle des web-cams et nous a posé une grande difficulté au départ. Pour cette raison, nous avons donc pendant un certain temps travaillé avec la webcam pour la réalisation de l'IHM (avant d'intégrer le programme de détection de couleur à l'IHM). Cela permettait d'avancer beaucoup plus vite sur le reste du code car on pouvait continuer le développement sur notre temps et nos ordinateurs personnels avec les webcams intégrées. De plus nous avions réussi à faire fonctionner la caméra uEye par ailleurs dans le programme de détection de couleur donc nous savions qu'il serait possible de le faire dans notre logiciel fusionné.

Cela nous a donc simplement rajouté une étape lors de l'intégration, mais cette dernire s'est finalement bien passé. La procédure d'ouverture de la communication avec la caméra et de réception des images doit être suivie très précisemment. En particulier, certaines valeurs obtenues lors de cette ouverture doivent être réutilisées à d'autres moments de la procédure et on ne peut donc pas découper la suite d'instruction comme on le veut. De plus, nous avons du mettre un temps d'inactivité sur ce thread entre chaque image car il récupère l'image puis l'envoie directement sans tenir compte de si le traitement de la précédente image est terminé ou non. Cela peut donc saturer l'ordinateur et l'interface se met à afficher une image qui est trop ancienne et le retard s'accumule.

Le fait que le banc sur lequel nos travaillions était modifié toutes les semaines: poussoirs qui disparaissaient et réapparaissaient, modification de la position de la caméra, de la lampe, etc, nous a fait perdre un temps assez considérable à chaque séance et a contribué au fait que notre projet ne soit pas totalement fini. Il fallait penser à prendre des photos du montage à chaque séance afin de pouvoir reproduire le plus fidèlement possible la disposition des éléments du montage.

Nous avions aussi un problème avec le tapis à chaque séance: il se mettait à trembler et à sauter alors qu'il fonctionnait bien la séance d'avant. Les premiers réflexes qu'il faut avoir lorsque l'on remarque une erreur avec le tapis sont les suivants: vérifier les branchements du milieu au niveau du connecteur carte spécifique-tapis (on se trompe rarement sur les connecteurs des bords mais il est très facile de faire une erreur de branchement des connecteurs du milieu), vérifier que l'on a mis une tension d'alimentation ni trop faible, ni trop forte au niveau de la carte spécifique.

Nous avions aussi rencontré un problème avec notre programme MBED ou notre carte Nucléo: le programme s'arrêtait de fonctionner automatiquement une ou deux minutes après avoir été lancé. Nous n'avons malheureusement pas eu le temps de nous pencher sur la question pour tenter de résoudre le problème.

3.2 Critiques sur les performances du prototype

Le prototype final que nous avons développé n'est malheureusement pas à la hauteur de nos attentes. Nous avons laissé trop de facteurs aléatoires pour assurer la robustesse minimale requise par notre système pour convenir que notre solution est un minimum fonctionnelle.

En effet, nous avions décidé de localiser la position de chaque cube à l'aide de la caméra (comme expliqué dans le paragraphe 2.1), mais la caméra n'était pas fixe, sa position changeait à chaque séance. Donc nous devions réajuster les temps de poussage à chaque fois que l'on revenait devant le montage. Aussi, soit nous avons mal choisi le moment où nous envoyions le signal de détection d'un cube à la carte Nucléo, soit nous avons mal disposé la caméra. Nous avons décidé de placer la caméra au début du tapis roulant et de d'envoyer l'information "couleur" du cube lorsqu'on le détectait pour la première fois. Mais, en faisant cela, nous n'obtenions jamais le même temps de poussée du cube, car cela dépendait de la façon dont on allait poser le cube sur le tapis: il fallait qu'elle soit invariante, ce qui est impossible. Nous aurions dû soit envoyer l'information une fois que le cube sortait du champ de vision de la caméra, soit laisser un espace assez grand pour poser le cube sur le tapis hors du champ de vision de la caméra. La dernière proposition comporte malheureusement un désavantage et c'est pour cela que nous avions "mal positionné la caméra" au départ: le premier poussoir est dans le champ de la caméra lorsqu'on la déplace trop loin. Sachant que ce poussoir était bleu, nous n'avons pas voulu prendre le risque d'avoir de mauvaises détections et avons préféré jouer la carte de la sûreté, une carte qui n'a malheureusement pas payé puisque nous ne sommes pas parvenus à faire fonctionner correctement le prototype.

De plus, le code que nous avons réalisé pour le tapis n'est pas optimal pour respecter le critère de rapidité défini dans le cahier des charges. Le fait d'attendre que le cube soit devant le poussoir pour l'actionner à l'aide d'une simple fonction wait, comme décrit au paragraphe 2.4.3, fait qu'il y a une période de temps pendant laquelle le programme est bloqué dans une boucle et ne peut rien faire. Cela peut provoquer des erreurs de tri si par exemple un cube est déposé pendant qu'il y a un 2ème cube non trié sur le tapis (à parti si sa couleur est verte, auquel cas il n'est pas poussé du tout) car décalage de temps dû au fait qu'on est en train d'attendre le moment pour pousser le premier cube. Pour régler ce problème, notre première idée était de faire un système dynamique avec des comparaisons de temps mais nous n'avons malheureusement pas réussi à le mettre en œuvre. Sinon, on pourrait aussi accélérer le tapis de sorte à ce qu'on ne puisse pas poser deux cubes en même temps, mais le système est limité par la vitesse des poussoir qui, malheureusement, n'est pas infinie donc nous pourrions nous retrouver limité.

Pour finir (quand même) sur une bonne note, ou pourra néanmoins constaté que notre algorithme de détection de couleurs ne rate aucune détection :)