Dimitri DECOMBE
Mathieu MOUGUES
Clément RAIGNER
Corentin NANNINI

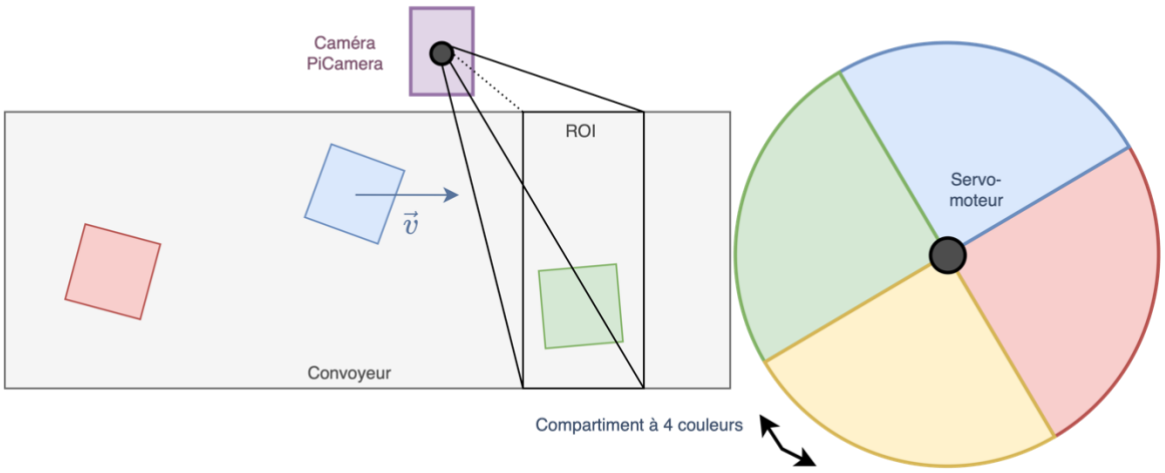
IOGS 2A Palaiseau

Projet Vision Industrielle

I. Description du projet

Le produit que l'on doit réaliser est un système de tri. Pour ce faire, nous disposons du matériel suivant :

- Un tapis convoyeur
- Des cubes de couleur



Notre système devra répondre aux exigences suivantes :

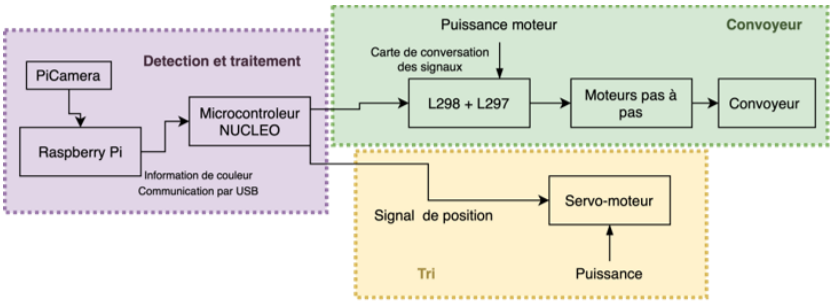
- Plus de **10 pièces/minute**
- Différencier le rouge/vert/jaune/bleu
- La **quantité de pièces triées** et le **temps moyen** de tri doivent être affichés
- Le système doit être **fiable :** moins d'une **erreur sur 1000 pièces**
- Interface **homme-machine** pour afficher la couleur des pièces

Pour atteindre ces objectifs, nous avons choisi d'utiliser :

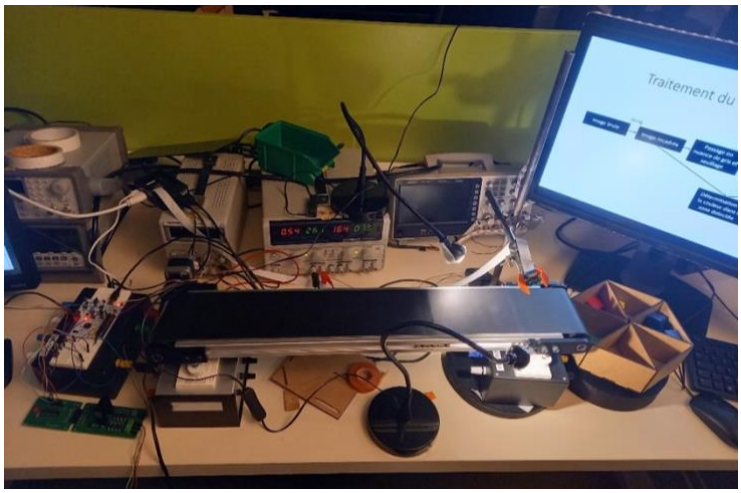
- Un servo-moteur qui asservit en position le bac de tri,
- Une caméra Raspberry PI pour traiter l'image et savoir la forme ou la couleur de l'objet sur le tapis,
- Une carte Arduino en guise de microcontrôleur,
- Un moteur pas à pas pour faire avancer le convoyeur
- Un bac de tri de forme hexagonale
- Un bac de tri de forme hexagonale où l'on peut trier jusqu'à 6 couleurs/formes différentes

Il faudra réaliser une liaison Raspberry Pi – Nucléo pour pouvoir réaliser le tri. Il s'agit de la partie la plus complexe du projet car c'est la partie où l'on a le moins de connaissances.

Toutes les fonctions codées ont été réalisées sur MBED. Nous présentons ci-dessous le schéma bloc de notre solution technique.



Voici à quoi ressemble notre système une fois fonctionnel :



II. Mise en œuvre d'une solution technique

1. Convoyeur

Dans cette partie nous voulons détailler la partie "convoyeur". Le convoyeur ou tapis roulant permettra aux pièces de couleurs de se déplacer et de les présenter sous la caméra.

Pour câbler le tapis roulant correctement, nous avons eu besoin d'une carte Nucléo, une carte de commande L298 et une carte de puissance L297 pour pouvoir contrôler le moteur pas à pas du tapis roulant.

Le principal enjeu de cette partie a été la connexion entre la nucléo et les cartes L297/L298. Les sorties du moteur ont été branché à la L298, elle-même connectée à la L297. Celle-ci est connecté à la nucléo par l'intermédiaire d'une *bread board.* Les sorties de la L297 sont branchées sur la sortie 3v3 de la Nucléo.

Nous avons eu quelques problèmes quant aux choix des caractéristiques d'alimentation pour le couple tension-courant et le rapport cyclique rc. En effet, le point de fonctionnement changeait à chaque séance.

Nous présentons ci-dessous, le code qui permettait de faire fonctionner le moteur pas à pas et qui a été implémenté dans la carte Nucléo.

Convoyeur.cpp

```cpp
#include "mbed.h"
// fichier pour faire tourner le convoyeur => pour l'instant tourne en continu
// Auteur: Mathieu MOURGUES
// Date: 07/02/2022

PwmOut moteur(D10);  // sortie pour contrôler la vitesse

int main() {
    int i, time = 1500;
    double rc;
    moteur.period_ms(1);
    rc=0.8;
    moteur.write(rc);

    while(1){
}
}
```

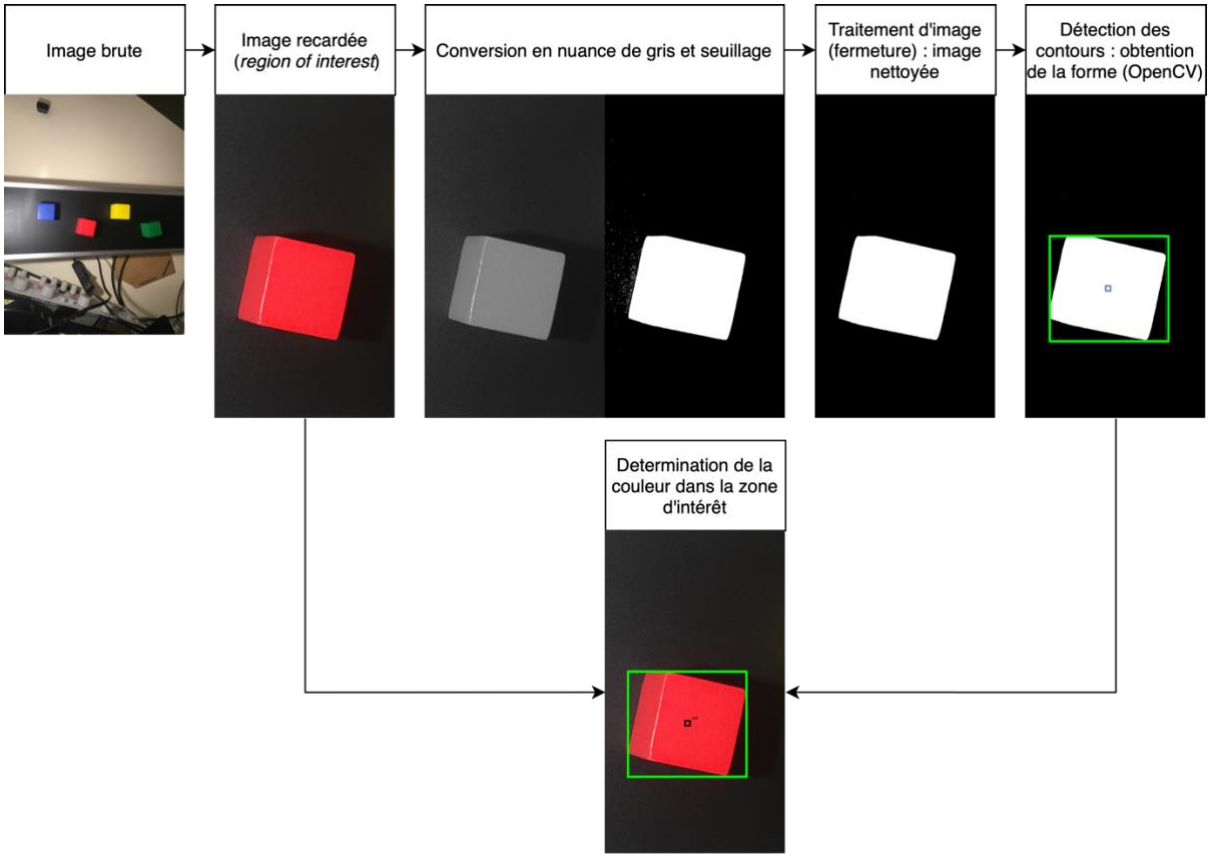
2. Traitements des images

Afin de détecter et de reconnaitre la couleur d'une pièce passant sur le convoyeur, nous utilisons un Raspberry PI 3 et un camera PiCam en utilisant la nappe de connexion dédiée.

Nous plaçons cette caméra au-dessus du convoyeur avec une potence, on peut positionner la caméra au plus proche du bac de tri afin de pouvoir avoir un espacement entre les pièces assez faible.

Les images sont acquises et traiter avec un scripts Python. Le code est disponible en annexe. Le code permet d'isoler une zone d'intérêt dans l'image et de pouvoir segmenter les pièces. Cette segmentation va permettre de déterminer la couleur moyenne au centre de la pièce (carré central).

La détection de la couleur permet alors d'envoyer une information sur le port série de la carte nucléo. Cela sous forme d'une chaine de caractères.



3. Tri des pièces

Maintenant que nous sommes capables de faire avancer les pièces grâce au convoyeur puis vérifier leur couleur, nous devons les trier. Pour ce faire, nous disposons d'un bac avec 6 compartiments en forme hexagone (comme on peut le voir sur la figure X). Il faut réussir à faire correspondre un compartiment à une couleur triée. Nous avons donc décidé de monter notre bac sur un servo-moteur afin de pouvoir piloter facilement son orientation en angle.

En effet, grâce à une commande PWM venant d'une carte nucléo, nous pouvons sélectionner l'angle de rotation du servo-moteur, et par conséquent, choisir un compartiment. Il suffit alors de choisir un angle pour chaque couleur que l'on souhaite trier. Ici, nous avons 4 couleurs à trier et un champ de rotation de 180°. Nous avons donc simplement calibré le servo-moteur pour réussir à attribuer un compartiment à une couleur précise.

Enfin, il faut réussir à asservir le bac de tri en angle en fonction de la couleur qui a été détectée sur le convoyeur. Pour cela, il faut notamment faire communiquer la carte nucléo contrôlant le servo-moteur et la Raspberry Pi. Nous avons opté pour une communication série entre les deux éléments que nous allons détaillée dans la partie suivante. On peut retrouver en annexe le code global sur la nucléo permettant de récupérer l'information de la Raspberry Pi et d'asservir le servo-moteur selon la couleur détectée. Le code correspondant est la fonction `move_trieur()`.

4. Liaison série Raspberry Pi – Nucléo

Il s'agit à présent de récupérer l'information de la caméra et de l'envoyer à la Nucléo pour pouvoir indiquer au servomoteur la rotation qu'il doit effectuer. Pour cela, on a procéder de la manière suivante :

- Du côté de la Raspberry Pi

Le système d'exploitation de la carte Raspberry PI est Linux. Il faut donc connaitre le nom du répertoire associé au port USB (universal serial bus) utilisé. Pour cela on importe la bibliothèque serial. Par la suite on déclare notre port série :

```
ser = serial.Serial(
    port = '/dev/ttyACM0',
    baudrate= 9600)
```

Cela nous permet alors d'envoyer des chaines de caractères avec la fonction `ser.write('')`.

- Du côté de la Nucléo

On a défini la liaison avec un RawSerial et le message que l'on reçoit est sous la forme string. On crée la fonction `readSerialPort()` qui nous permet de déterminer s'il faut lire l'information envoyée par la Raspberry Pi.

On utilise la fonction `pi.baud(9600)` qui signifie que la liaison série permet de transmettre au maximum 9600 bits/seconde. On utilise aussi la fonction `pi.attach()` qui permet de lancer la fonction à l'intérieur dès que la connexion série s'interrompt via le `readSerialPort()`.

III. Bilan de l'équipe

A la fin, le projet est fonctionnel et l'on est capable de trier des pièces très proches les unes des autres, qu'elles soient centrées ou non sur le tapis. Les couleurs pouvant être triées sont le rouge, le vert, le bleu et le jaune.

Les points de perfectionnement de notre système sont :

- Pouvoir trier plus de couleurs : de ce point de vue, on est principalement limité par le bac de tri qui ne comporte que 6 compartiments dont seulement 4 sont utilisables avec le servomoteur. Du côté du traitement de la couleur, il faut juste rajouter les niveaux de couleur.
- Pouvoir trier des formes : il s'agit ici principalement de devoir modifier la forme de notre détecteur de contour au niveau du tri de l'information.

On peut donc se dire qu'il serait assez aisé de rajouter des formes et des couleurs ou même les deux ensembles.

- Réaliser une interface H/M : par manque de temps, on n'a pas pu s'y consacrer. Ce projet est assez difficile à quantifier au niveau du temps à y consacrer car on n'en a jamais réalisé. Nos choix techniques ne seraient peut-être pas les meilleurs pour réaliser une interface via MatLab par exemple.

Au niveau de la cohésion d'équipe, tout s'est bien déroulé et les compétences de chacun ont été utilisées. On a réussi à tenir le planning prévu jusqu'à l'avant-dernière séance où l'on a dû travailler sur la liaison série Raspberry Pi – Nucléo. Cette partie-là nous a pris plus de temps que prévu et l'on a même eu peur de ne pas réussir à temps.

Pour conclure, le projet s'est très bien passé et l'on peut trier des objets selon les performances attendues. Quelques améliorations peuvent être apportées mais elles ne modifieraient en rien le fonctionnement du système en lui-même.

Annexe

Code sous Python pour Raspberry Pi

```
import cv2
import random
import numpy
import picamera
import picamera.array
import time
import io
from fractions import Fraction
from picamera.array import PiRGBArray
from picamera import PiCamera
import os
import serial

# Déclaration du port série USB pour communiquer avec la nucleo
ser = serial.Serial(
    port = '/dev/ttyACM0',
    baudrate= 9600
)

# La fonction de detection de couleur
def couleurRVB(triplet):
    #input : un triplet BGR
    #output : un char correspondant a la couleur
    # déclaration message output pour Nucléo
    msg = "none"

    b_mean = triplet[2]
    g_mean = triplet[1]
    r_mean = triplet[0]
    # displaying the most prominent color

    if (b_mean > g_mean and b_mean > r_mean):
        msg="b"
        ser.write(b'b') #envoie sur le port série
    if (g_mean > r_mean and g_mean > b_mean):
        msg="g"
        ser.write(b'g')
    if (r_mean > g_mean and r_mean > b_mean):
        msg="r"
        ser.write(b'r')
    if  (r_mean > b_mean and g_mean > b_mean and abs(r_mean-g_mean) < 0.2): #si rouge et vert proche
et supérieur au bleu alors jaune > définir seuil
        msg="y"
        ser.write(b'y')
    return msg

# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 10
camera.exposure_mode = 'off'
camera.awb_mode = 'off'
camera.awb_gains = (1.5,1.7)

camera.brightness = 50
```

```
### If the image is too dark of to bright
#change those parameter
#shutter speed
camera.shutter_speed = 50000
#ISO : sensibility
camera.iso = 400

rawCapture = PiRGBArray(camera, size=(640, 480))
# allow the camera to warmup
time.sleep(0.1)
text2 = ''
# capture frames from the camera
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
    # grab the raw NumPy array representing the image, then initialize the timestamp
    # and occupied/unoccupied text
    image =cv2.rotate(frame.array[:,214:427,:], cv2.ROTATE_90_CLOCKWISE).copy()
    image = cv2.convertScaleAbs(image, alpha = 1, beta = 0)
    key = cv2.waitKey(1) & 0xFF
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).copy()

    _, threshold = cv2.threshold(gray, 15, 255, cv2.THRESH_BINARY)

    kernel = numpy.ones((12,12), numpy.uint8)
    #Morphologie mathematique : Erosion
    morpho = cv2.erode(threshold, kernel, iterations = 1)
    #Fermeture
    contours, _ = cv2.findContours(morpho, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    i = 0

    for contour in contours:
        # the all image is detected as a shape
        # filtrer les petits parazites
        if cv2.contourArea(contour) < 1000:
            continue
        #taille du carré où sera moyenné les pixels pour la couleur finale
        sizerect = 5
        #Création d'un rectange entourant le sujet
        x,y,w,h = cv2.boundingRect(contour)
        cv2.rectangle(image, (x, y), (w+x, h+y), (0,255,0), 2)

        #image qui continet la couleur au centre de la zone et surlignage de la zone
        sousimg = image[y + h//2-sizerect : y+h//2+sizerect,x+w//2-
sizerect:x+w//2+sizerect,:].copy()
        cv2.rectangle(image, (x+w//2-sizerect, y + h//2-sizerect), (x+w//2+sizerect, y +
h//2+sizerect), (255,255,0), 2)
        #extraction des composantes
        b = numpy.sum(sousimg[:,:,0])
        g = numpy.sum(sousimg[:,:,1])
        r = numpy.sum(sousimg[:,:,2])
        Max = max((r,g,b))
        r = r/Max
        g = g/Max
        b = b/Max
        #affichage de la couleur
        text = "["+str(r)+","+str(g)+","+str(b)+"]"
        text2=couleurRVB((r,g,b))
        cv2.putText(image,text2,(int(x + w//2),int( y + h//2 )), cv2.FONT_HERSHEY_DUPLEX,0.5,
(255,255,255), 1)
        cv2.putText(image,text,(int(x + w//2),int( y + h//2 )+15), cv2.FONT_HERSHEY_DUPLEX,0.5,
(255,255,255), 1)

        cv2.putText(image,"last color : "+text2,(15,15), cv2.FONT_HERSHEY_DUPLEX,0.5, (255,255,255),
1)

        #affichage de l'image finale avec les rectangles
    cv2.imshow('FINAL', image)
    if text2 == 'y':
        time.sleep(1.6)
    rawCapture.truncate(0)

cv2.destroyAllWindows()
```

Code sous MBED pour Raspberry Pi

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

#include "mbed.h"
#include "platform/mbed_thread.h"
#include <string>
#include <RawSerial.h>

// Initialisation des différents ports de la Nucléo
DigitalOut led(LED1);
RawSerial  pi(USBTX, USBRX);
PwmOut servo_mot(D9);
PwmOut moteur(D10);       // sortie pour contrôler la vitesse
InterruptIn mybutton(USER_BUTTON);

// Définition des variables
char msg;                // variable pour récupérer les infos depuis la
                           Raspberry Pi
double rc;               // variable pour le rapport cyclique du tapis

//servo_mot.pulsewidth_us(1000);   // Initialisation en position 0

void format (){
    }
void move_trieur(void);
void pressed(void);                // Déclaration de la fonction
                                     d'interruption 1
void pressed(){                    // fonction d'interruption 1
        if( msg == 'r'){
            msg = 'b';
            }
        else{
            msg = 'r';
            }              // permet de décaler la valeur binaire de 3 bits
                          vers la droite - équivalent à une division par 2^3
    move_trieur();
}

// Fonction pour déterminer si on doit lire ou pas l'information de la Raspberry Pi
void readSerialPort() {
  msg = ' ';
  if (pi.readable()) {
    msg=pi.getc();
  }
}


// Fonction principale pour déterminer de combien doit tourner le servomoteur pour le bac
de tri
void move_trieur(){
  if (msg == 'r') {
    servo_mot.pulsewidth_us(1000); //orientation de la boite pour le
                                     compartiment rouge
  }
  else if(msg=='g'){
    servo_mot.pulsewidth_us(1);    //orientation de la boite pour le
                                     compartiment vert
  }
  else if(msg=='b'){
    servo_mot.pulsewidth_us(2000); //orientation de la boite pour le
                                     compartiment bleu
```

```
  }
  else if(msg=='y'){
    servo_mot.pulsewidth_us(1500); //orientation de la boite pour le
                                    compartiment jaune
      }
  else {
                                   // cas non géré
  }
}

void loop() {
  readSerialPort();
  move_trieur();
  }

int main() {
    servo_mot.period_ms(20);         // Initialisation période
    pi.baud(9600);

// Passage pour faire tourner le tapis
    moteur.period_ms(1);
    rc=0.8;
    moteur.write(rc);
    pi.attach(&loop, Serial::RxIrq);
    wait(3.5);

    while(1){
        }
}
```