

Jimeng Zhou
Marie Fondanèche
Maimouna Diao
Malcolm Chatony

RAPPORT TECHNIQUE - VISION INDUSTRIELLE



Table des matières

INTRODUCTION.....	2
Objectifs.....	2
Schéma de principe.....	2
Notice d'utilisation.....	3
Cahier des charges - contraintes et performances attendues.....	4
Schéma fonctionnel.....	4
DESCRIPTION.....	5
Rotation du tapis.....	5
Rotation du trieur.....	5
Initialisation de la caméra.....	6
Traitement d'image.....	7
Transfert des données via RS232.....	9
Interface IHM.....	10
Intégration.....	11
Bilan.....	11
Partie technique / Avancement final.....	11
Retour d'expérience de l'équipe (comparaison Gantt.....)	12

INTRODUCTION

Objectifs

Notre équipe a pour objectif de créer un système de tapis roulant permettant de trier les objets qui circulent dessus en fonction de leur couleur. Ils seront ensuite dirigés dans des boîtes différentes, selon une répartition déterminée par l'utilisateur.

Le système sera accompagné d'une interface permettant de choisir cette répartition (*ie.* quelle couleur va dans quelle boîte), et pourra tourner en continu (pas d'arrêt du tapis).

- Le trieur utilisé est un trieur circulaire tournant compartimenté, capable de présenter au bout du tapis la boîte voulue. Il tourne grâce à un moteur continu.
- Le tapis tourne grâce à un moteur pas à pas, à une vitesse constante et sans interruption.
- La caméra sera munie d'une lampe annulaire permettant un éclairage uniforme de la cible. Elle est connectée à l'ordinateur via un câble USB.

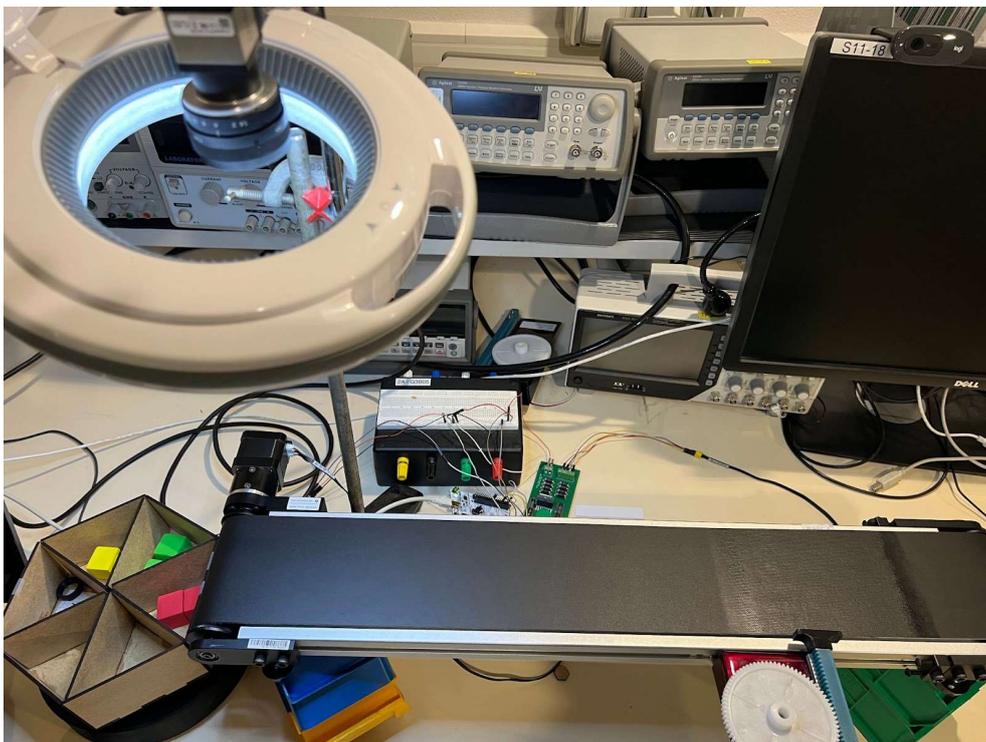


Image descriptive du montage

Schéma de principe

Les matériels à notre disposition sont

- une caméra de détection de présence de l'objet et de couleur
- un éclairage stable et homogène pour la détection de couleur
- un moteur pas à pas associé au tapis roulant qui transporte les objets à trier

- un moteur à courant continu associé à la boîte de triage
- une carte Nucleo pour le codage du pilotage des deux moteurs
- une carte L297 et une carte L298 pour l'alimentation et le fonctionnement du moteur tapis

Voici notre schéma de principe qui regroupe tous les éléments dans notre montage :

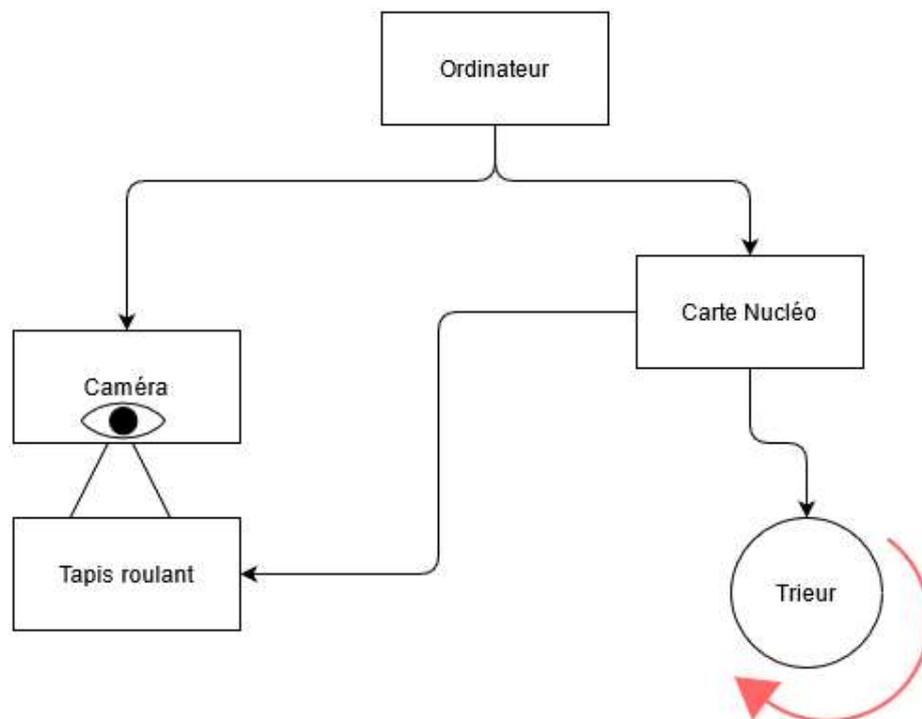


Schéma de principe du système

Notice d'utilisation

À faire une fois le câblage effectué et le code téléversé sur la carte Nucléo. Le tapis tourne déjà, et une répartition par défaut des couleurs est déjà définie (les couleurs sont séparées dans les 4 boîtes). En l'absence d'interface reliée avec le code, il faut changer la répartition directement dans le code python. Il suffit ensuite de déposer les objets à trier sur le tapis. Il est important de séparer les objets de 3,2 secondes afin de ne pas avoir d'erreur.

Remarque : si le lien interface-code avait été réussi, on aurait pu choisir à tout instant de l'expérience une répartition en faisant une sélection à la souris.

Cahier des charges - contraintes et performances attendues

Nous avons quatre objectifs à atteindre à la fin du projet :

- **Rapidité** : La fréquence de détection du système est au minimum 10 pièces/min.

- **Détection** : Le système doit pouvoir détecter, différencier et trier dans les différentes boîtes 4 couleurs de base (rouge, vert, jaune et bleu). Il faut aussi afficher la quantité de pièces triées et le temps moyen par pièce.
- **Fiabilité** : Sur la détection des couleurs de base, une erreur d'une pièce sur 1000 est tolérée.
- **Ergonomie** : Une interface humain-machine est envisagée, ce qui permet de transmettre la couleur (ou la forme) des pièces à trier. Elle doit être utilisée sans formation préalable.

Schéma fonctionnel

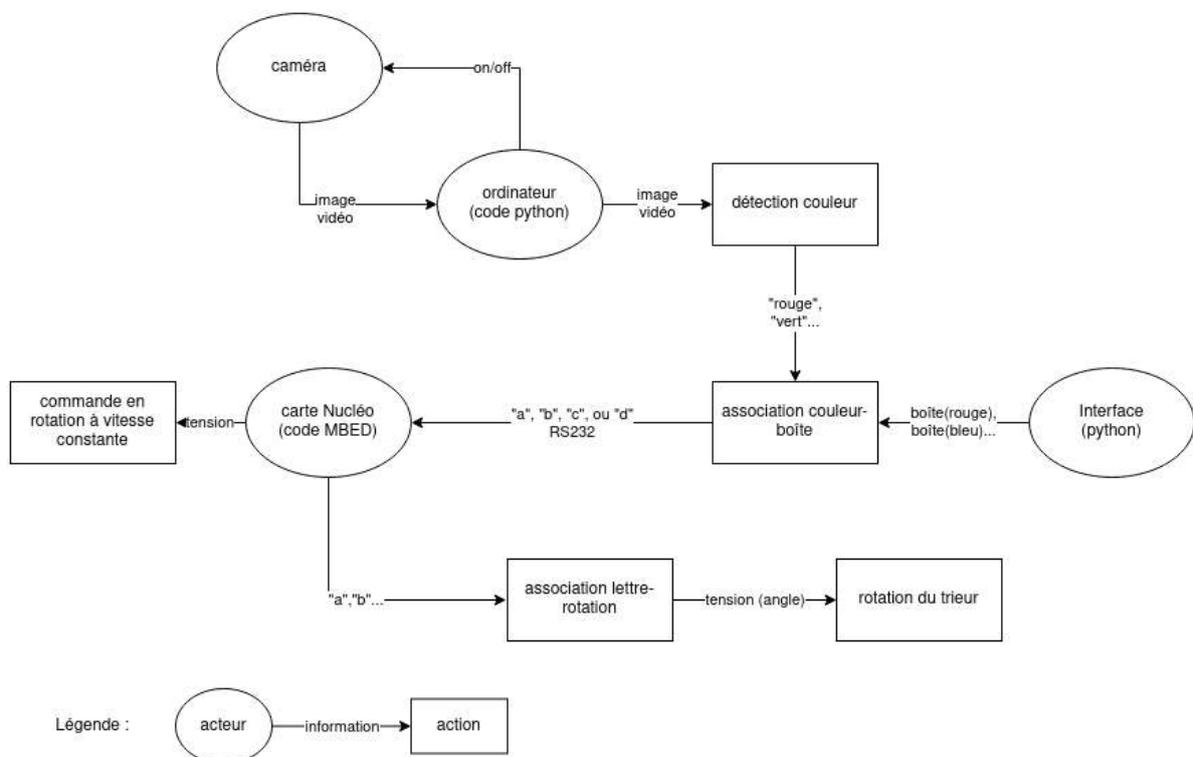


Schéma fonctionnel du système envisagé

Nous avons opté pour une présentation avec les éléments physiques du montage encadrés, les actions encadrées, et les informations échangées sur les flèches.

DESCRIPTION

Rotation du tapis

Dans cette partie, l'objectif est de faire rouler le tapis avec une vitesse constante dans un sens de rotation définie.

Nous avons à notre disposition une carte Nucléo qui est capable de fournir une tension de 5V, une carte d'extension L297 et L298 avec un étage de puissance basé sur une structure de pont en H.

Pendant le câblage qui relie le moteur avec les cartes d'extension, nous avons rencontré certaines difficultés. Sur le moteur déjà monté dans le tapis roulant, nous n'avons pas trouvé de référence. Et sur les sites internet, nous n'avons pas pu trouver de fiche technique sur le fonctionnement du moteur.

Dans ce cas, nous avons décidé de collaborer avec un autre groupe qui travaille sur le même type de moteur. Après le bon câblage, nous avons codé sur MBED afin de choisir et déterminer la période, le rapport cyclique et le sens de rotation.

Pour l'étape d'essai, nous n'avons pas pu réussir dès le premier coup, ce qui nous a permis d'apprendre une technique de débogage : l'utilisation de l'oscilloscope. A l'aide du câble coaxial et l'affichage de l'oscilloscope, nous pouvons visualiser le signal à l'endroit quelconque dans le circuit électronique du montage. En observant et analysant les signaux, nous pouvons détecter l'endroit où vient le problème.

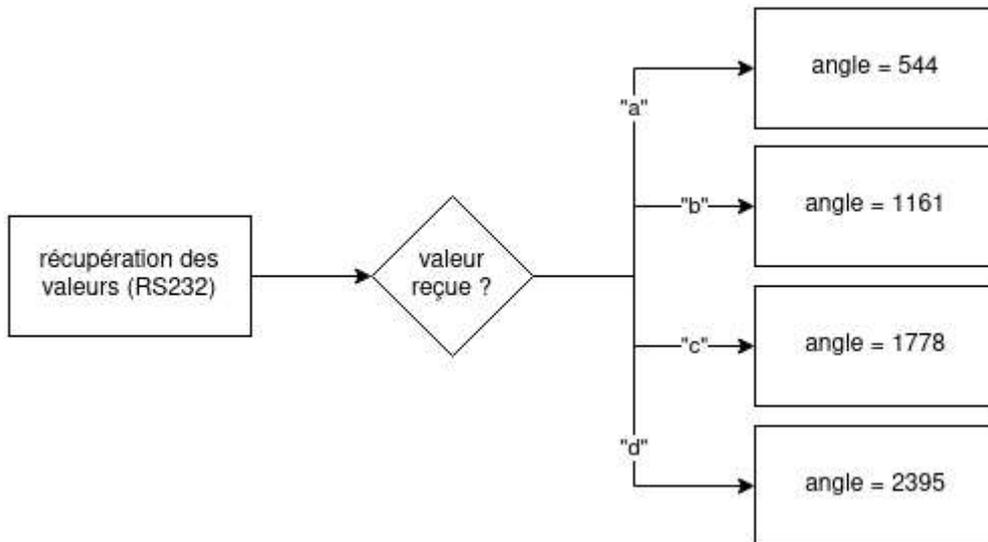
Finalement, nous avons trouvé que les masses des cartes n'ont pas été reliées ensemble. Nous avons pris la leçon de vérifier d'abord les câblages de base et de bien ranger et positionner les câbles et les cartes afin d'éviter de se perdre dans la situation délicate.

A la fin, le tapis est capable de tourner avec une vitesse constante et dans le sens défini par utilisateur. Le temps parcouru par un objet à trier entre les deux extrémités du tapis est d'environ 30 secondes.

Rotation du trieur

Notre objectif était de faire tourner le trieur de sorte à ce que le compartiment choisi par l'utilisateur pour chaque couleur s'aligne avec le tapis en fonction de la couleur détectée.

Pour ce faire, nous utilisons une carte Nucléo. Elle reçoit en entrée en continu depuis l'ordinateur une information sur le compartiment à présenter devant le tapis : "a" correspond au premier compartiment, "b" au second, "c" au troisième, "d" au quatrième. La rotation adéquate pour chacun de ces angles est simplement une commande en PWM vers le moteur à courant continu du trieur, avec une largeur de pulse différente pour chaque angle : XX pour le premier, XX, pour le second, XX pour le troisième et XX pour le quatrième.



Algorithme de détermination de la rotation du trieur

Afin de valider notre code et le câblage effectué, nous avons marqué chaque compartiment, et avons décidé manuellement sur MBED d'orientations du trieur. Le trieur s'orientant correctement, nous l'avons ensuite lié à l'ordinateur via la liaison RS232 et avons choisi une répartition sur l'ordinateur, en ajoutant au code une partie permettant de récupérer l'information venant de l'ordinateur et assignant l'orientation à effectuer directement au trieur.

À ce moment du projet, la reconnaissance de couleur marchait déjà en direct et nous avons pu immédiatement tester avec des cubes de couleur. Nous avons remarqué que si deux cubes de couleur étaient trop proches, de sorte que le premier cube n'avait pas le temps de tomber dans le trieur avant que le suivant ne soit détecté, alors le trieur tournait trop tôt et les deux cubes atterissaient dans le dernier compartiment.

La fréquence maximale de traitement de cube est donc l'inverse du temps de parcours des cubes de la caméra au trieur. Nous avons réduit cette distance au minimum et avons atteint un traitement d'une vingtaine de cubes par minute.

Initialisation de la caméra

Une des premières tâches était de commencer par la prise en main de la caméra IDS couleur dont nous disposons sur notre montage.

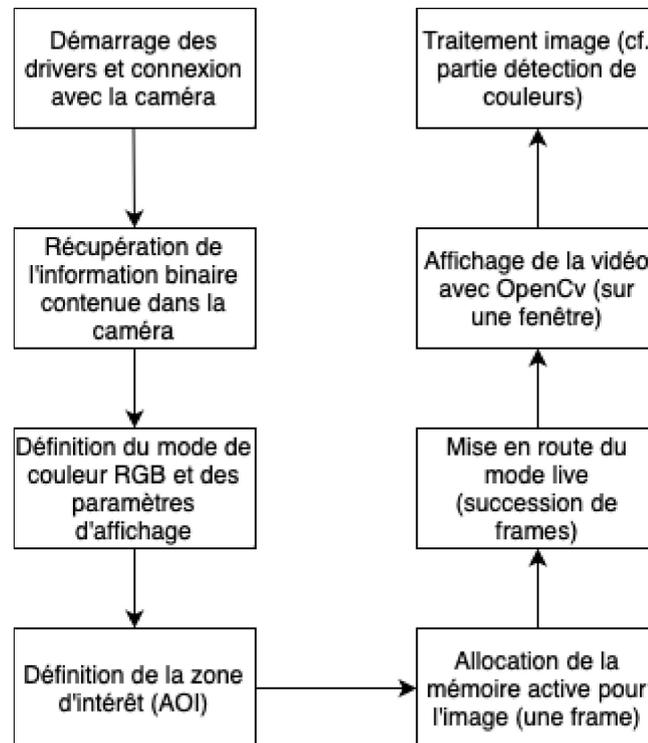
En effet, nous avons déjà eu l'idée d'utiliser python pour commander la caméra car la simplicité de la syntaxe python et le fait qu'il s'agisse d'un langage d'assez haut niveau avec des bibliothèques très avancées et pertinentes pour nos applications en faisait une solution très adaptée.

Au départ on s'était donné comme objectif de prendre une photo au passage de chaque cube ou d'en prendre plusieurs à intervalle régulier et court afin de les stocker puis de les analyser dans la partie de traitement d'image. Mais en cherchant sur internet, on s'est vite rendu compte qu'aucune source fiable ne nous permettait d'arriver à nos fins.

C'est pourquoi on s'est plutôt basé sur le code source fourni par le constructeur IDS afin de réaliser à la fois, l'allumage de la caméra, une initialisation et l'affichage de ce qui est

capté par le capteur sur une fenêtre d'affichage "cv" sur l'ordinateur. Cette solution présente plusieurs avantages. Une fiabilité garantie, l'utilisation de bibliothèques communes avec le traitement d'image et surtout, une vision en continu des cubes qui avancent sur le tapis, ce qui permet une grande robustesse du prototype.

Après avoir remanié quelques détails du code fourni nous sommes capables de résumer le fonctionnement du code sous l'algorithme suivant :



Algorithme de mise en route de la caméra

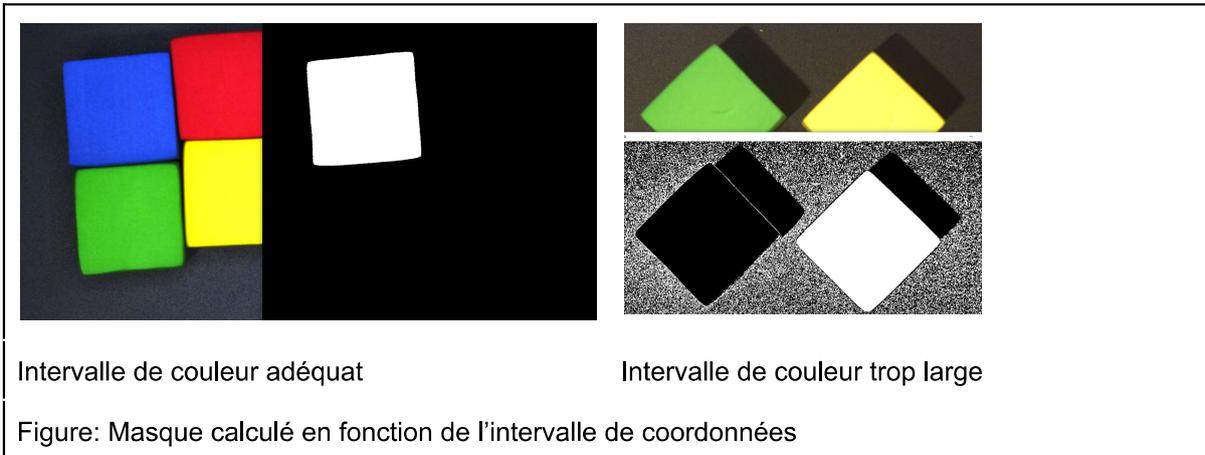
La validation fut assez directe car il suffisait de connecter la caméra sur l'ordinateur et de vérifier que la fenêtre d'affichage opencv s'ouvrait bien, pour quitter l'affichage sur la fenêtre **OpenCv** et arrêter la caméra il suffit d'appuyer sur le bouton "Q" sur la clavier de l'ordinateur.

Traitement d'image

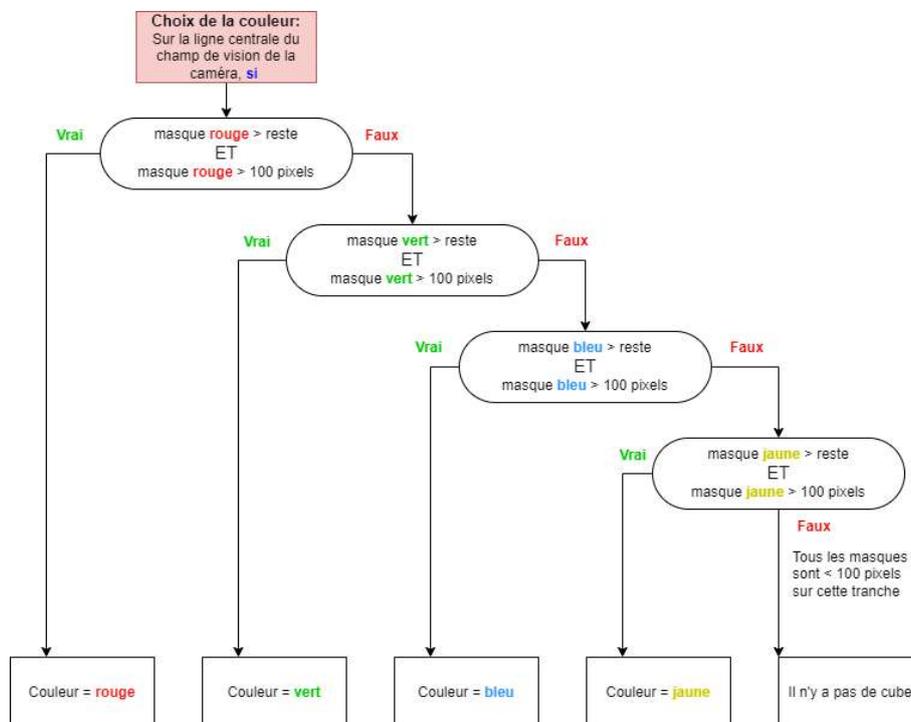
Cette partie traite principalement de la reconnaissance des cubes et de leur couleur par la machine. Pour ce faire, nous avons utilisé la bibliothèque **Opencv** de python. Le code établi fonctionne dans un environnement de mesure très particulier. En effet, dès que la luminosité change, la caméra voit de toutes autres couleurs et peut donc ne pas détecter les cubes. Pour cause, pour que les cubes soient reconnues, on crée des intervalles de coordonnées colorimétriques de format **HSV** pour chaque couleur. Pour chaque image, le code crée un masque (une image binaire) où les pixels dont la couleur appartient à cet intervalle sont blanc et tout le reste est noir.

Il est légitime de se demander: "Pourquoi ne pas agrandir ces intervalles pour être sûr que la couleur soit détectée?". En faisant cela, on laisse place à beaucoup de bruit, ce qui rend

notre mesure beaucoup moins précise. Pour fixer les conditions de mesure, on choisit d'ouvrir la caméra à **N=4** et d'utiliser la lampe fournit à son **intensité minimale**.



Une fois ces intervalles bien définis, nous voulons que le code puisse nous dire “on détecte un cube de telle couleur” ou “on ne détecte pas de cube”. Pour ce faire, on réalise un seuillage. Dans un premier temps, on demande à l’algorithme de compter le nombre de pixel blanc dans le masque pour établir si un cube est présent dans le champ de vision de la caméra ou non. On se rend vite compte que cela prend du temps et comme les mesures sont faites en temps réel, cela n’est pas adéquat. On demande alors de compter, pour le masque de chaque couleur, le **nombre de pixel blanc se trouvant sur la ligne médiane verticale**. Le côté d’un cube représente environ 300 pixels. Le protocole suivant définit alors le choix de la couleur:



Algorithme de la détection du cube et du choix de la couleur du cube

Cette calibration permet de mettre n'importe qu'elle fond tant qu'elle n'a pas une couleur similaire à l'un des cubes.

A ce stade, nous attribuons arbitrairement une boîte à chaque couleur. Plus tard, lors de la phase dédiée à l'interface Homme Machine, nous allons tenter de demander à l'utilisateur de choisir lui-même la couleur qu'il souhaite mettre dans chacune des boîtes. Dans la partie sur MBED, les boîtes étant nommées par les variables 'a','b','c','d', nous envoyons celles-ci via le protocole RS232. Lorsqu'il ne détecte aucun cube, la variable 'e' est renvoyée. Celle-ci n'a aucune incidence sur le reste du code. Enfin, pour stopper la partie de détection de couleurs indépendamment de l'affichage de la vidéo de la caméra sur l'ordinateur, il suffit d'appuyer sur 'k'.

Transfert des données via RS232

Une fonction importante de notre projet consiste à envoyer les instructions de l'utilisateur sur le rangement des couleurs issu de la partie sur le traitement image. En effet, une fois une couleur détectée, un caractère de type 'char' : a,b,c,d qui sera envoyé par liaison USB RS232 afin que ce dernier soit interprété par le trieur de cubes. Quant à la correspondance entre couleurs et boîtes, il en retournera de l'interface.

On commence par se renseigner sur la liaison RS232 et on découvre qu'il s'agit d'un protocole de communication qui est compatible avec les langages C (type Mbed) et python. Ce protocole est très intéressant pour nous car il permet de communiquer de l'information du pôle python vers le pôle Mbed, mais également dans l'autre sens. On a un canal RX pour l'émission et un canal TX pour la réception.

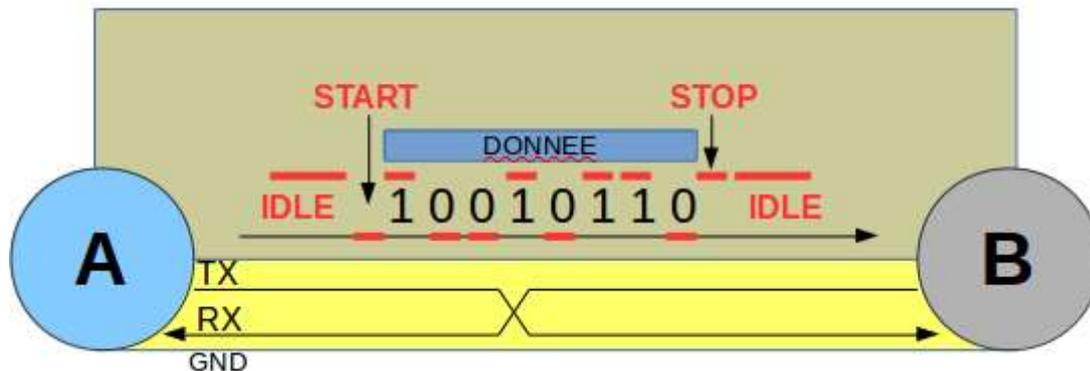


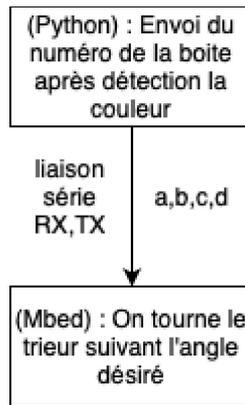
Schéma de principe du protocole RS232, source :

<http://lense.institutoptique.fr/mine/nucleo-configurer-une-communication-point-a-point-de-type-rs232-2>

∟

Nous avons trouvé aussi intéressant le fait qu'on puisse communiquer en asynchrone avec la liaison RS232. Cela nous a permis d'utiliser une méthode de communication par interruption sur la réception du port série, ce qui permet de réaliser une action (i.e tourner le trieur) dès qu'une donnée apparaît sur le canal.

Voici un algorithme explicatif de notre code sur python et Mbed :



Principe de la communication entre nos deux pôles via RS232

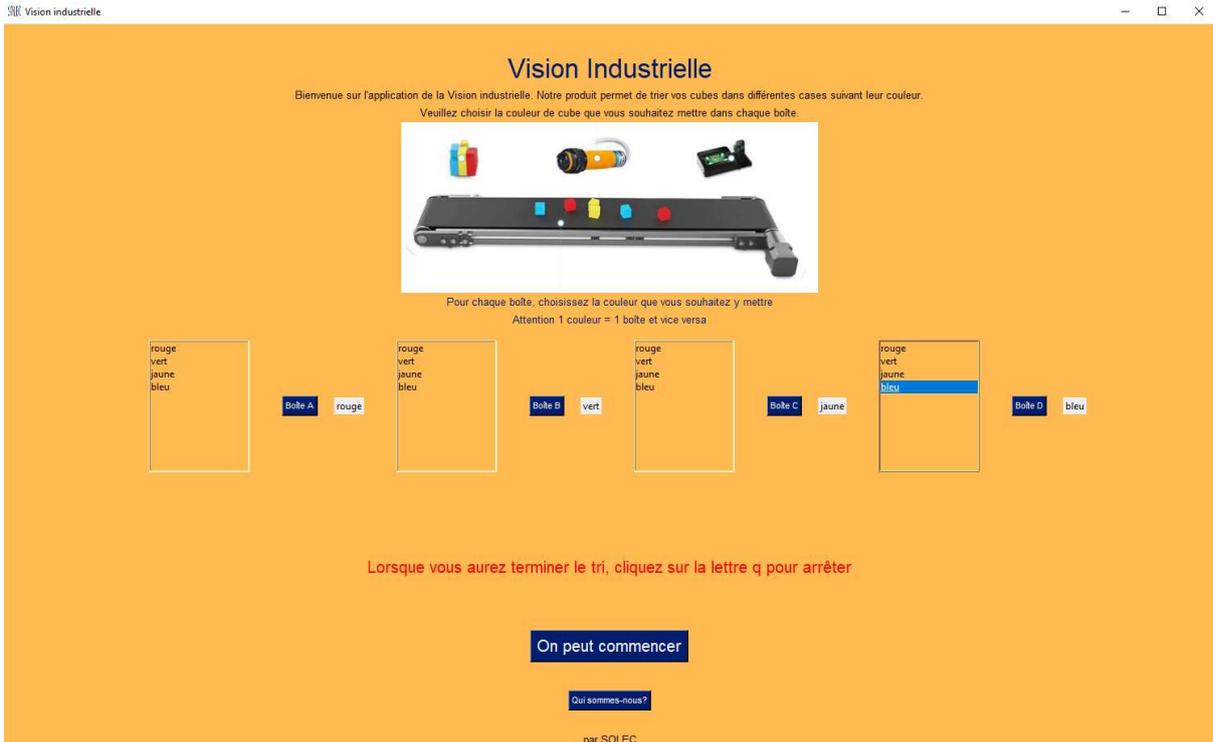
On note une attention particulière au bon branchement sur le port USB RX et TX de l'ordinateur en utilisant la fonction 'UnbufferedSerial'. De plus, il est nécessaire de bien paramétrer la vitesse d'échange de données entre le pôle python et le pôle Mbed afin que la communication puisse avoir lieu, ici on a choisi 115200 bauds, soit 115200 bits/s.

Pour la validation de notre code, nous avons commencé par réaliser un test simple l'allumage de la led intégrée d'une carte Nucléo, lors de l'envoi de la lettre 'a' on veut que la led s'allume, au contraire si l'on envoie 'e', elle doit s'éteindre. Cela fut un succès, puis vint le moment de mettre en commun avec la partie traitement d'image où la détection d'une couleur, le rouge par exemple, fait allumer la led de la Nucléo. Après cette deuxième validation, on a donc décidé de garder cette méthode de communication avec comme format le 'char' pour l'envoi de données ('a','b','c','d' pour le numéro de boîte). Cela permet d'être plus simple à interpréter avec les fonctions read sous Mbed.

Interface IHM

L'interface a pour but de faciliter l'usage du montage. Elle est codée sur python à l'aide des outils de la bibliothèque **Tkinter**. Elle permet de présenter brièvement le montage et demande à l'utilisateur de choisir la couleur que l'on souhaite mettre dans chaque boîte. Pour améliorer l'ergonomie, on utilise les fonctions de personnalisation de la librairie. On ajoute également un bouton permettant de commencer l'acquisition dès que le choix des couleurs est fait.

Pour l'attribution des couleurs à chaque boîte, nous utilisons la fonction **Listbox** permettant de créer des listes puis un bouton pour valider le choix. Ainsi, la personne clique, par exemple, sur "rouge" dans la liste, puis sur le bouton "boîte A" pour valider. Le choix de la personne est noté à côté pour confirmation. Cette partie fonctionne bien: la bonne valeur est affichée. Cependant, la variable contenant la couleur existe uniquement au sein de la fonction dans laquelle elle est appelée et ne peut pas être utilisée dans la suite. Nous créons donc une variable globale pour y pallier mais pour une raison qui nous est inconnue, cela n'a pas fonctionné: la variable n'existe plus dès que l'on sort de sa fonction. Nous avons essayé de trouver des alternatives telles que l'usage des checkbox mais par faute de temps, cela n'a pas pu aboutir. On a finalement gardé la répartition boîte-couleur faite par défaut.



Nous aurions aimé également rendre le choix boîte-couleur bijectif: attribuer à chaque couleur une unique boîte et vice-versa.

Intégration

L'intégration des différents codes s'est faite au fur et à mesure, ce qui l'a énormément facilité. Du côté de MBED, pour le tapis et le trieur, il n'y a eu aucun souci de concaténation des codes. Du côté du pôle python, le code de traitement d'image a été fait à part. Le code d'usage de la caméra est une boucle infinie dans laquelle une partie est dédiée à rajouter un code de traitement.

Lors de la mise en commun, il y a eu un problème de lenteur car, comme stipulé ci-dessus, la reconnaissance du cube était faite en traitant l'ensemble des matrices de masques de couleur. C'est à ce moment que nous avons décidé de ne traiter qu'une seule ligne de la matrice. Cela a résolu immédiatement le problème. Ensuite, l'étape de l'intégration du protocole RS232 s'est faite dans le code MBED, avec assez d'aisance. Pour la partie python, nous avons eu un problème parce que le code RS232 contient une boucle infinie, tout comme celui de la caméra et cela ne fait pas bon ménage. Nous avons cependant réussi à modifier le code de telle sorte à ce qu'elle sorte de la boucle infinie du protocole à chaque itération.

Le dernier problème d'intégration auquel nous avons fait face est celui de l'interface. Nous n'avons malheureusement pas trouvé de solution dans le temps imparti.

Bilan

Partie technique / Avancement final

Enfin, en ce qui concerne notre planning de départ, toutes les tâches assignées, à l'exception de l'interface, ont été réalisées à temps, que ce soit les plus longues ou les plus courtes. Nous avons donc validé toutes les tâches que nous nous étions assignés sur notre diagramme de Gantt.

En effet, dès le début, on a choisi de se séparer en pôles suivant les préférences et compétences de chacun.e afin de mieux s'organiser et les tâches des 2 pôles étaient gérées de manière indépendantes et en parallèle: un pôle python (composé de Malcolm et Maimouna) et un pôle MBED (composé de Jimeng et Marie). Dans chaque pôle, chaque membre testait son code seul.e afin de vérifier qu'il fonctionne correctement puis, les 2 membres mettaient leurs codes en commun. A la fin du projet, on n'avait qu'à tout assembler et réaliser les tests techniques. Nous avons eu donc très peu de problèmes de "variables" au sein du code.

Retour d'expérience de l'équipe (comparaison Gantt...)

Pour finir, nous établirons ici un bilan sur notre ressenti en tant qu'équipe de notre projet et dresserons une comparaison entre nos prévisions et la finalité réelle de notre projet.

Tout d'abord, chaque membre de l'équipe se sent content.e de l'avancement final du projet malgré les difficultés rencontrées. Nous avons chacun.e trouvé que l'organisation sur Notion était très appropriée et que nous avons une bonne entente entre nous, cela nous a permis d'être unis dans la difficulté et de se motiver mieux pour arriver à nos fins.

Le seul regret que nous avons eu c'est de ne pas avoir pu aller jusqu'au bout de nos attentes en ce qui concerne la finalisation de l'interface car on aurait tous aimé finir le projet dans son intégralité.

