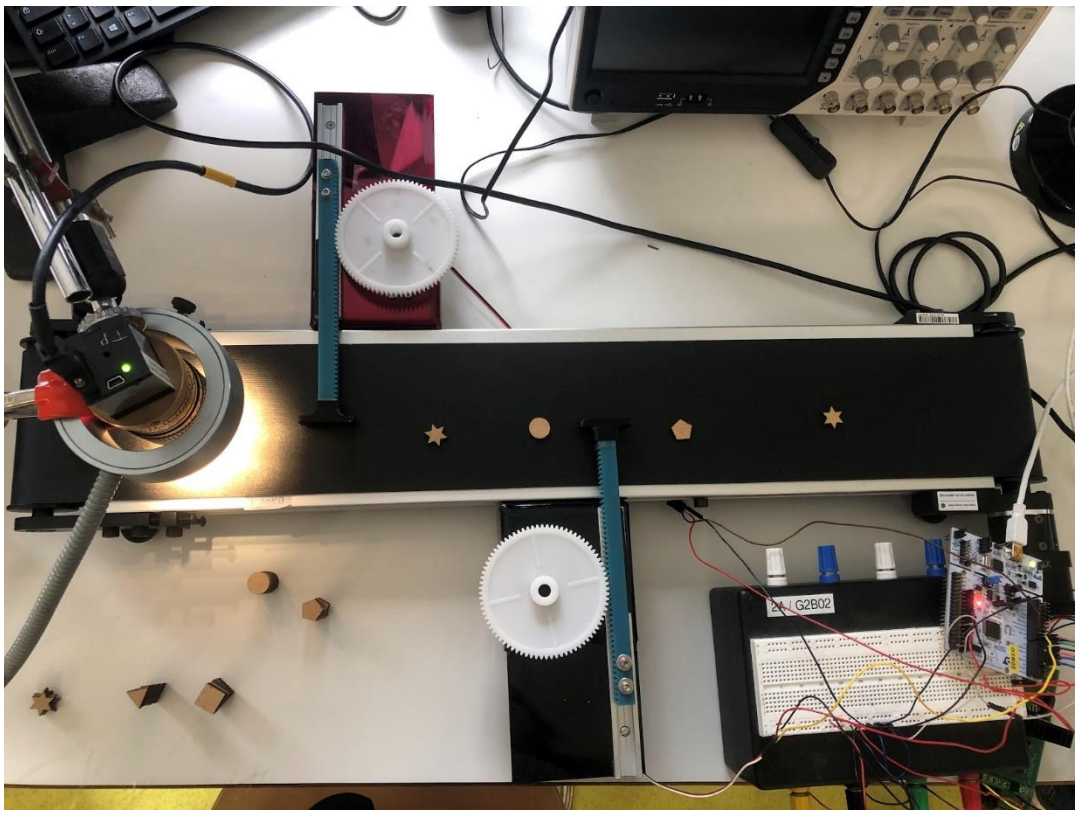
Vision industrielle 2022-2023



[MARS 2023]

[solec&co]
Auteur.trice.s : [Léa BRITO, Simon MILCENT, Clara SONCIN, Lucile TILLOY]

Table des matières

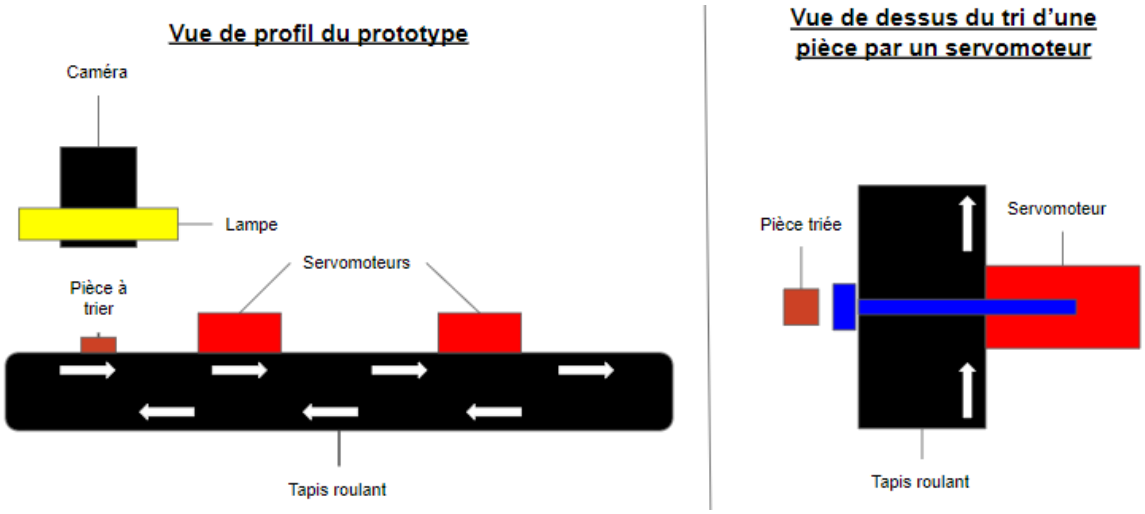
Table des matières

Introduction

Nous avons lors des dernière semaines effectuées un projet pour la société Solec. Le but de notre projet était de trier des formes sur un tapis roulant. Tous les moteurs sont commandés à l'aide d'une carte Nucléo, la caméra est contrôlée sous Python et l'interface est codée sous Python également.

Schéma principe

Le principe du montage est de permettre de trier des pièces amenées par un tapis roulant à l'aide de servomoteurs et d'une caméra. La caméra détecte la pièce à trier à l'entrée du tapis roulant et sa forme est déterminée grâce à un traitement d'image. Une fois la forme de la pièce déterminée, un signal associé à celle-ci est envoyé à la carte Nucléo. En fonction des formes que l'utilisateur a choisi de trier avec chaque servomoteur, la carte Nucléo déclenche le mouvement du servomoteur, qui est relié à un bras permettant de pousser la pièce voulue hors du tapis roulant. Si le signal envoyé à la carte Nucléo ne correspond pas aux formes que l'utilisateur a choisi de trier, aucun servomoteur n'est activé. Le tapis roulant tourne en continu pour apporter les pièces.



Cahier des charges /Contraintes et performances attendues

- Trier une dizaine de pièce par minute
- Trier des formes différentes→ Au moins 4 différentes
- Une erreur d'une pièce sur 1000 est tolérée

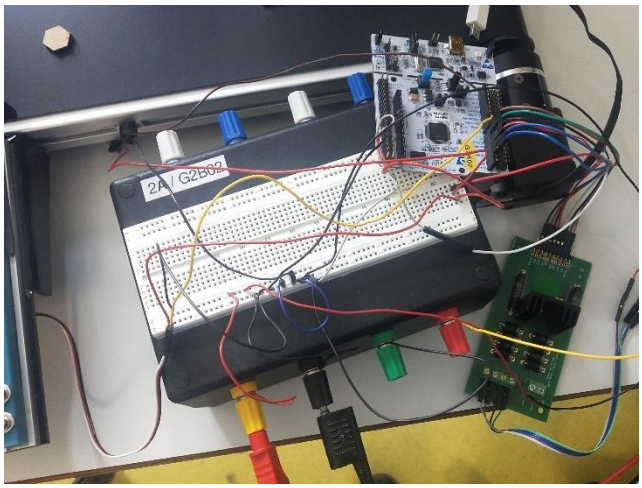
On a choisi de ne pas s'occuper des couleurs des pièces car on a utilisé une caméra monochromatique.

Notice d'utilisation

On donne ici la notice d'utilisation (du moins l'initialisation) des différentes parties du système indépendamment. La mise en commun est à la fin de cette notice.

Démarrage des moteurs :

Relier le moteur pas-à-pas du tapis et les servomoteurs à la carte Nucléo, en utilisant notamment le driver double pont Effectuer les bons branchements (se référer au code Mbed pour avoir accès au bon port de communication avec la carte Nucléo), la photo ci-dessous montre la complexité des branchements, il est conseillé pour quelqu'un qui récupèrerait le projet de bien analyser le code Mbed avant de se lancer dans ceux-ci. Le tapis doit être alimenté avec une source externe à 5V et les servomoteurs peuvent être alimentés directement à la sortie 5V de la carte Nucléo. Une fois la mise en place faite télécharger le code sur la carte et activer la source. Les servomoteurs devraient s'initialiser à une position zéro décidé ici pour être au milieu du tapis et le tapis devrait avancer.



Si le tapis ne roule pas bien, d'abord essayer d'inverser deux câbles de commande. S'il ne roule pas dans le bon sens, inverser tous les câbles de commandes ou inverser le sens directement sur Mbed.

Démarrage de la caméra :

Allumer la lumière entourant la caméra pour assurer un bon traitement d'image. Pour allumer la caméra, il faut lancer le système Python et vérifier qu'aucun message d'erreurs n'apparaissent (sur l'initialisation de la caméra et sur l'allocation de mémoire). Si cela est le cas, relancer le logiciel Python ou l'ordinateur si cela ne fonctionne toujours pas. Si aucun message d'erreur apparaît, la caméra est en fonctionnement. Le logiciel va afficher pour chaque prise d'image de la caméra : le nombre de forme détectés et la lettre correspondant à la forme la plus grande de l'image.

Activation de l'interface :

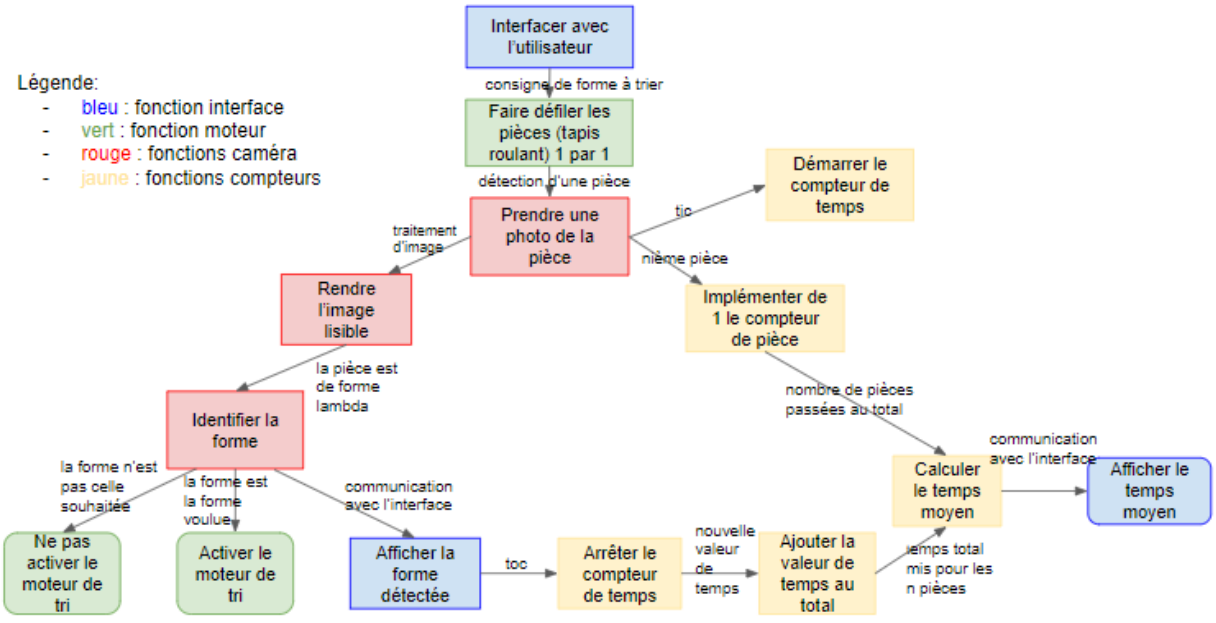
Pour activer l'interface, il faut lancer le script associé, soit "interface4.py". Une fois le script lancé, une fenêtre apparait dans la barre des tâches de l'écran, il s'agit de la fenêtre de l'interface. A cause des problèmes de versions Python incompatibles sur les ordinateurs, nous n'avons pas pu relier l'interface

au projet final, l'interface fonctionne donc indépendamment du reste du projet. Pour tester le bon fonctionnement de l'interface et notamment l'utilisation des boutons, il suffit de cliquer sur l'un d'eux sur la fenêtre de l'interface. Un texte apparait alors dans la Command Window de Spyder. Ce texte est différent pour chaque bouton, par exemple, pour le bouton "Arrêter le tri", le texte apparaissant dans la Command Window est "Le test arrêt fonctionne". En appuyant sur le bouton "Trier les formes", une deuxième fenêtre s'ouvre. Celle-ci fait également partie de l'interface, elle devait initialement permettre de choisir quelle forme trier avec quel servomoteur, mais comme la mise en commun n'a pas été finalisée, cocher l'une des formes n'est associé à aucune fonction.

Mise en commun :

Lorsque toutes les fonctionnalités sont prêtes, vérifier que les moteurs sont alimentés et que les tests caméras soient corrects et lancer le tapis. Déposer ensuite une série de pièces de formes différentes sur le tapis et vérifier que les servomoteurs s'activent au bon moment. Pour le code choisi pour le moment, il y a deux servomoteurs qui sont utilisés : si la caméra détecte un carré, c'est le deuxième servomoteur qui va s'activer, si on détecte un triangle c'est le premier servomoteur qui s'active. Pour arrêter le système, il faut que celui-ci détecte une étoile à 6 branches. Ainsi pour couper le prototype, il faut mettre des pièces en forme d'étoile.

Schéma fonctionnel



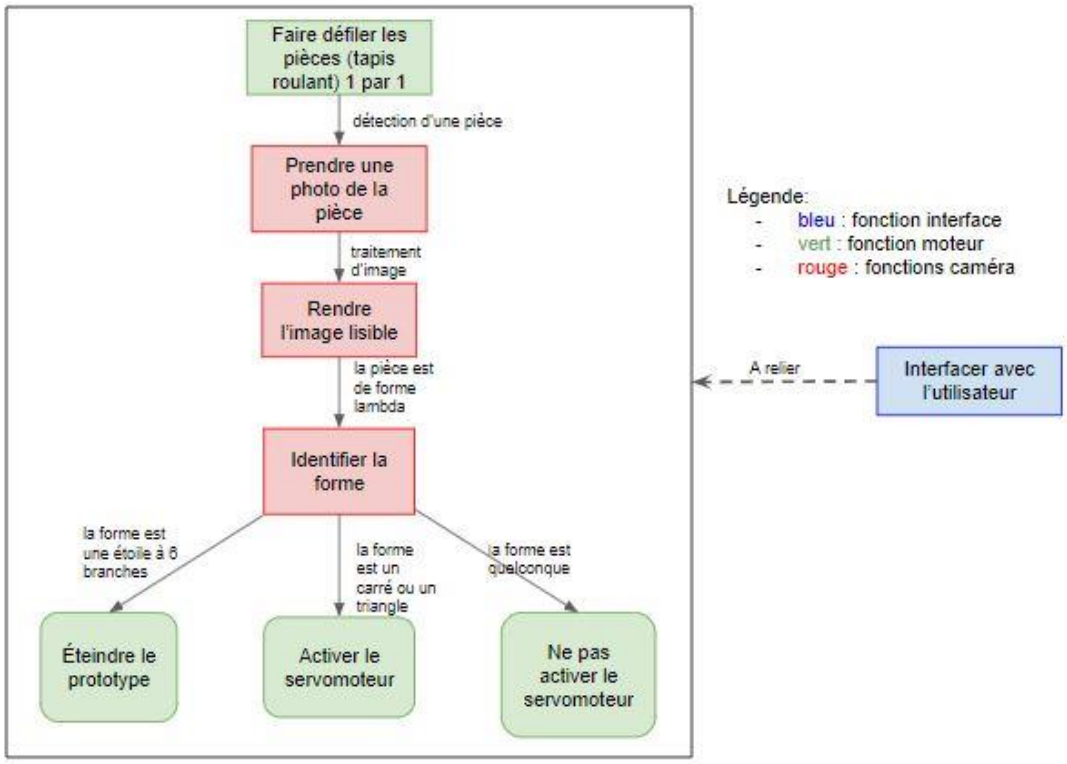
Dans ce schéma fonctionnel, seule la partie de gauche a été mise en place. Pour le moment, l'interface n'a pas encore été mise en lien avec le reste du système. De plus, on n'a pas encore mis en place les

mesures de temps pour le moment. Toutes les autres fonctions marchent indépendamment. On a ainsi pour ce prototypes les fonctions principales suivantes :

- Le contrôles des moteurs : le tapis roulant et les servomoteurs.
- Le contrôle de la caméra et le traitement d'image
- Une interface fonctionnelle (à adapter au système)
- Communication et lien entre les différents éléments.

Fonctionnement du prototype

Schéma de fonctionnement



Pour développer ce projet, chaque étape a d'abord été réalisée indépendamment les unes des autres avant que la mise en commun de toutes les fonctionnalités soit faite.

Tapis roulant

Le tapis fonctionne grâce à un moteur pas à pas contrôlé par un driver double pont (carte L298) Le code se fait en C++ et active les unes après les autres des bobines qui font tourner le moteur.
Lors du branchement des composants du tapis, il faut bien faire attention à ce que les bobines soient branchées au bon endroit sinon la transition se fait mal et le tapis ne roulera pas.

Algorithme :

1. Attribution des bobines à une commande
2. Activation des commandes tour à tour, chaque activation de commande est suivie d'un petit temps d'attente.
3. Répétition du processus jusqu'à coupure du courant

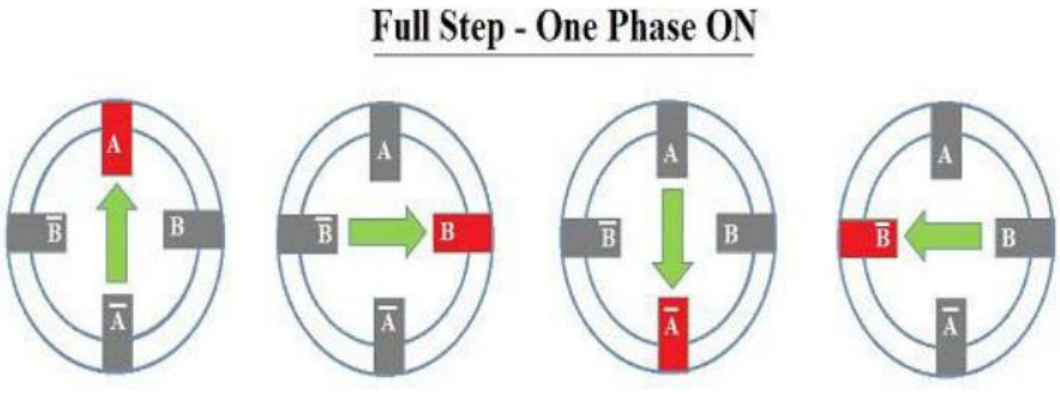
Test de fonctionnement : Les premiers branchements des bobines se font aléatoirement, il est ainsi possible que les bobines ne fonctionnent pas de façon alternée. Pour vérifier si l'on a affaire à ce problème, inverser deux fils voisins pour vérifier ce qu'il se passe

Servomoteur

Le servomoteur fonctionne grâce à un moteur angulaire, contrôler par une carte Nucléo, les connexions se font à l'aide d'une alimentation à 5V, une masse et un câble de commande. Après avoir initialisé le zéro, il suffit d'indiquer une nouvelle position pour faire bouger le moteur.

Algorithme :

1) Initialisation de l'angle
2) Si le signal d'activation est reçu
   a) Rotation d'un angle positif
   b) Attente
   c) Rotation d'un angle négatif



Communication entre la carte Nucléo et Python

Pour activer le servomoteur au bon moment, l'ordinateur doit envoyer un signal au bon moment, pour ce faire on effectue une communication RS232.

Il est important d'indiquer sur Python un port de commande, une vitesse et faire de même sur le code permettant de contrôler la carte Nucléo.

Ensuite on peut envisager que selon le résultat reçu la carte effectue une action. Dans notre cas les servomoteurs sont actionnés quand une lettre est reçue. Il suffit donc pour Python d'envoyer la lettre quand le bon moment est venu, c'est-à-dire quand la forme à trier a été détectée

Algorithme :

1) Connexion de la carte Nucléo sur les bonnes variable et choix de la vitesse de communication
2) Recherche des ports de communication de Python ainsi que choix de la vitesse, attention les deux vitesses doivent être les mêmes.
3) Envoie d'un signal par Python
4) Réception du signal par la carte Nucléo et renvoie d'un signal de confirmation
5) Réception du signal de confirmation (comme une lettre recommandée avec avis de réception)
6) Fermeture du port de communication

Test de fonctionnement : Pour tester la communication sur la carte Arduino, l'utilisation de TeraTerm est efficace, on peut envoyer des commandes pour qu'une LED s'allume ou s'éteigne selon ce qui est envoyé sur TeraTerms.
Une fois cette étape passée, les complications sur python ne sont que formelles.

Traitement d'image à l'aide de la caméra

On utilise une caméra Ueye monochrome pour pouvoir détecter les formes des objets circulant sur le tapis roulant. Le but de cette fonction est de pouvoir identifier les différentes formes de objets et de renvoyer une lettre spécifique associée à chaque forme pour pouvoir la réutiliser pour l'activation ou non des servomoteurs.

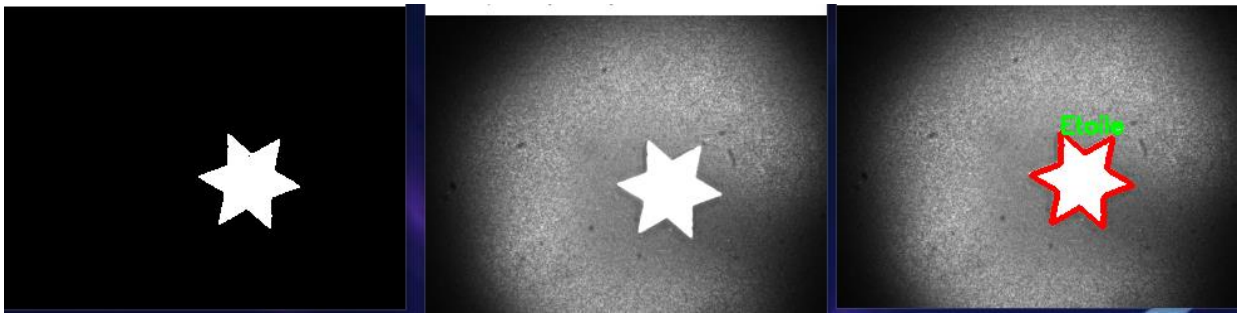
On fait le traitement de la caméra et le traitement d'image sous Python en utilisant principalement deux bibliothèques : la bibliothèque Ueye pour la caméra et la bibliothèque Opencv pour faire le traitement d'image.

Pour le contrôle lui-même de la caméra, il est important d'initialiser celle-ci, de lui allouer un espace mémoire et surtout, de le vider de bien fermer la caméra après l'utilisation. Si cela n'est pas fait, la caméra va se lier à une autre application de l'ordinateur et on ne pourra plus y avoir accès. Dans ce cas, le seul moyen d'en récupérer le contrôle est d'éteindre l'application Python ou parfois l'ordinateur lui-même. Il faut donc éviter cette erreur pour être plus efficace. La caméra fonctionne alors en continue et enregistre des images à intervalles régulier, que le logiciel Python va ouvrir et traiter.

Pour la partie traitement d'image, nous avons utilisé un seuillage ainsi que les fonctions érosion et dilatation de la bibliothèque Opencv afin d'avoir une forme bien nette à identifier. Le seuillage permet de binariser l'image : les zones plus lumineuses vont s'afficher en blanc et les zones plus sombres en

noirs. On fait ensuite une érosion de cette image, le système va alors donner à tous les pixels la valeur du pixel minimale à tous les pixels de cette zone. La dilatation fait l'opération inverse en donnant la valeur maximale. Ces deux opérations à la suite permettent d'obtenir une image plus nette, en réduisant les réflexions parasites et les défauts des pièces. Nous utilisons ensuite la fonction contours pour pouvoir identifier la forme à l'aide du nombre de côtés que la fonction renvoie. Pour être sûres de ne pas être embêter par les réflexions parasites qui resteraient même après le traitement d'image, nous avons écrit un petit algorithme permettant de ne garder que le plus gros contour détecté. Une fois le contour identifié, une lettre est associée à la forme détectée et on renvoie la lettre correspondante.

Test de fonctionnement : Pour tester cette fonction, nous avons renvoyée l'image que nous avions après le traitement d'image (on pouvait voir le seuillage) et une image sur laquelle nous mettions en évidence la forme du contour détectée par le programme.



Nous avons un exemple ci-dessus : au milieu, c'est l'image captée par la caméra, à gauche, l'image issus du traitement (après le seuillage, à l'érosion et la dilatation), à droite, on dessine la forme que le logiciel détecte et son nom afin de vérifier la concordance avec la réalité.

Algorithme :

1) Initialisation de la caméra et de l'espace mémoire
2) Prise d'une image par la caméra
3) Traitement de l'image, dans l'ordre : seuillage, érosion et dilatation
4) Repérer les formes et extraire le nombre de contours de la plus grande forme
5) En déduire la forme exacte et envoyé la lettre correspondante à la carte nucléo
6) Reprendre à 1 jusqu'à l'arrêt du système

Interface (non reliée au reste du montage)

L'interface utilisateur a été réalisée avec la bibliothèque QTDesigner de Python et le logiciel associé. La fenêtre principale de l'interface a d'abord été créée sur le logiciel de designer, et a été enregistrée dans un fichier spécial appelé fenetre.ui . La fenêtre secondaire a été réalisée de la même façon. Le code Python "interface4.py" permet de faire le lien entre le fichier des fenêtres, les boutons de l'interface et des fonctions associées à chaque bouton.

Algorithme :

1) Lis le fichier de la fenêtre principale

2) Associe à chaque bouton de l'interface un nom utilisable dans le code Python

3) Associe à des actions sur les boutons une fonction sous Python

4) Affiche l'interface

5) Définie les fonctions précédentes

6) Lis le fichier de la fenêtre secondaire

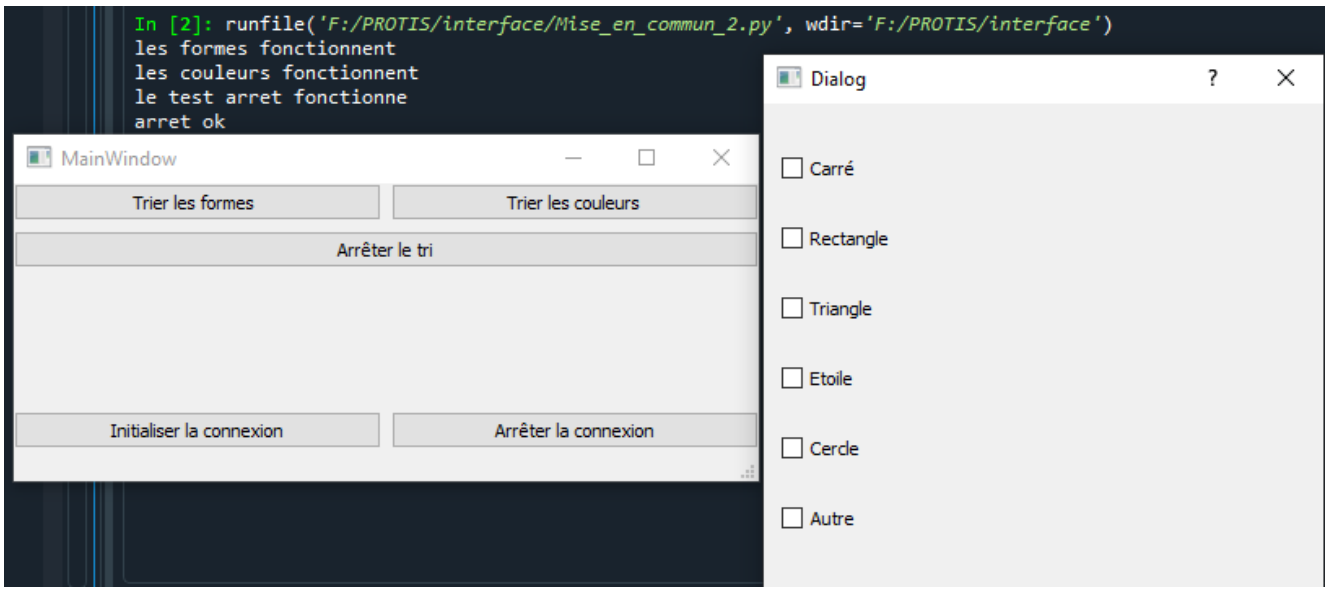
7) Associe un nom à chaque boîte à cocher

8) Affiche la fenêtre secondaire

Test de fonctionnement :

Pour tester le bon fonctionnement de l'interface, il suffit de lancer le code Python associé à celle-ci. Il faut cependant que le code soit dans le même dossier que le fichier définissant les fenêtres de l'interface. Une fois que l'interface est apparue, il suffit de cliquer sur un des boutons et d'observer la Command Window du logiciel Python pour vérifier qu'un message apparait. Pour le bouton "Trier les formes" il faut également vérifier que la seconde fenêtre de l'interface apparait également.

Images de l'interface et des messages dans la Command Window:



Mise en commun des fonctions

La dernière étape est la mise en commun de tous ces éléments. Pour ce faire, il faut faire attention à ce que le tapis roulant continue de fonctionner lorsque le servomoteur reçoit le signal de faire un aller-retour et à ce que le système puisse détecter plusieurs formes en même temps.

Algorithme :

1) Prise d'une image par la caméra
2) Si la forme détectée n'est pas celle demandée
   a) Retour au point 1
3) Sinon envoie du signal pour activer le bon servomoteur
   a) Envoie du signal de transmission entre python et la carte Nucléo
   b) Envoie de la Confirmation de réception du message de la carte à Python
   c) Prise en main du tapis roulant de façon indépendante jusqu'à ce que la pièce atteigne le bon servomoteur
   d) Aller-retour du servomoteur
4) Retour au point 1

Nb : le tapis fonctionne de façon continue et doit être actionné de façon indépendante seulement lors de l'actionnement des servomoteurs.

Conclusion

Avancement final

Comme il a été mentionné pour le schéma fonctionnel, les fonctions principales sont mises en place indépendamment : on peut contrôler les moteurs et donc le tapis et les servomoteurs, on peut détecter des formes grâce à la caméra et une interface a été construite. Une communication a été mise en place entre la caméra et les servomoteurs grâce à une connexion RS232 et par le biais de lettre : le logiciel de contrôle de la caméra envoie des lettres correspondant aux formes à la carte Nucléo. Le code Mbed a alors associé certaines lettres à certaines actions. L'interface n'est pas encore connectée au reste du système, mais elle est prête pour que cela soit le cas : il suffit de lier les codes effectués à celle-ci.

Au niveau du cahier des charges, on respecte le critère sur la diversité des formes détectées : on détecte actuellement les formes suivantes : un carré, un rectangle, un triangle, un pentagone, une étoile à 6 branches. Toutes les autres formes vont être considérés comme un cercle. On est donc à 6 formes détectés. Il est cependant facile de détecter de nouvelles formes : il suffit d'associer un nombre de côtés à une forme dans le programme python. Pour les autres critères, on ne les a pas testés comme on n'a pas programmé la gestion du temps dans ce prototype, mais il semble que cela ne soit pas le cas : l'image captée va être légèrement déformée car les formes sont déplacées par le tapis. Ce phénomène crée des erreurs de lecture de formes qui sont accentué par le traitement d'image. En effet, les fonctions érosion et dilatation ont un effet secondaire : ils vont lisser les bords des formes,

notamment les coins. En conséquence, certains coins peuvent être considérés comme des côtés de l'image. Ainsi des triangles peuvent être vu comme des carrés ou des rectangles. Ce qu'il faut donc faire par la suite c'est adapté la vitesse du tapis et améliorer le traitement d'image pour être plus fiable.

Un autre problème qu'on a rencontré était que l'image perçue et le traitement, notamment le seuillage, dépendaient trop de l'éclairage extérieur. Ainsi, on devait reprendre le traitement à chaque début de séance. Pour contrer cela, on doit rendre l'éclairage indépendant de l'extérieur : on ajoute donc une lampe qui va éclairer les pièces. Ceci rend le système plus robuste aux changements d'environnement.

Ainsi par la suite, on pourrait apporter les vérifications et améliorations suivantes pour perfectionner le prototype :

- Vérifier que l'éclairage mis en place permet bien une robustesse du système à l'éclairage de l'environnement
- Mettre en place l'interface
- Etudier le temps d'exécution de chaque action en temps réel
- Rendre le traitement par la caméra fiable même pendant un mouvement

Lorsque le prototype sera stable et fiable, la prochaine étape est de remplacer la caméra pour pouvoir ensuite étudier en parallèle la couleur des pièces.

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

Quatre personnes ont travaillé ensemble sur ce projet. Pour partager les informations sur les différents aspects du projet, on a utilisé un drive partagé, sur lequel chaque fonction avait son dossier. Cela permettait ainsi de se partager les sources, les difficultés, les solutions trouvés et les points importants à savoir pour la communication entre les différents programmes.

Pour l'organisation elle-même, on a séparé le travail par fonction : une personne s'est chargée de l'interface, une du contrôle des moteurs et un binôme du contrôle de la caméra et du traitement d'image. Le but était ensuite de se charger tous ensemble de la communication, comme chacun de nous connaissait une partie différente du projet. Pour avoir le temps de faire cela, on a réalisé au début du projet un diagramme de Grant.

	Séance 1		Séance 2		Séance 3		Séance 4		Séance 5 (audit)	Séance 6
Découverte										
Moteurs										
Interface utilisateur										
Caméra										
Connexions des fonctions										
Améliorations										
Finalisation										

	Simon
	Léa
	Lucile
	Clara
	Tous

Voici ci-dessus notre diagramme initial. On espérait finir assez tôt afin d'avoir le temps de régler les problèmes éventuels de connexions et pour faire des améliorations. On s'est cependant rendu compte que nos fonctions individuelles étaient plus complexes à faire que prévu et la connexion elle-même à poser plus de difficultés que prévu car les différents ordinateurs ne disposaient pas de la même version de Python. Au final, on a suivi le planning suivant :

	Séance 1		Séance 2		Séance 3		Séance 4		Séance 5 (audit)	Séance 6
Découverte										
Moteurs										
Interface utilisateur										
Caméra										
Connexions des fonctions										
Améliorations										
Finalisation										

	Simon
	Léa
	Lucile
	Clara
	Tous

Avec ce projet, on a donc appris que le planning était une étape cruciale. On n'a ici pas assez planifié et surtout pas assez pris en compte les différents problèmes qu'on aurait pu rencontrer. En conséquence, nous avons en retard sur notre programme.

Mis à part ce problème, cette séparation du travail a permis à ce que chacun se spécialise dans un domaine. Chacun de nous connait parfaitement le code et les enjeux de la fonction sur laquelle il ou

elle a travaillé, et a une vision globale du reste du travail. Cela permettait aussi, en cas de problème, de demander de l'aide à quelqu'un qui a un point de vue extérieur à la situation et donc de prendre du recul. En conséquence, on pouvait voir de petites erreurs, qu'on ne voit pas forcément lorsqu'on est plongé dans le même problème pendant un certain temps. De plus, chacun de nous a ses faiblesses et ses points forts, chacun a donc pu apporter une contribution à ce projet. Nous en avons tous ressorti des compétences techniques mais aussi relationnelles, que ça soit par la communication ou l'entraide.

Annexe

Code Mbed

```
/* mbed Microcontroller Library
 * Copyright (c) 2019 ARM Limited
```

```
 * SPDX-License-Identifier: Apache-2.0
 */

#include "mbed.h"
/// Variable tapis///
DigitalOut Bob1(D2);
DigitalOut Bob2(D3);
DigitalOut Bob3(D4);
DigitalOut Bob4(D5);
DigitalOut EnaA(D7);
DigitalOut EnaB(D8);

/// Variable Servo_moteur///
int attente_servomot_1=32000;
int attente_servomot_2=32000;
PwmOut servo_mot_1(D9);
PwmOut servo_mot_2(D10);
InterruptIn mybutton(PC_13); // Déclaration de l'interruption
//DigitalOut myled(LED1);

///Variable Comunication RS232///
DigitalOut led1(LED1);
UnbufferedSerial      my_pc(USBTX, USBRX);
char data;
void ISR_my_pc_reception(void);



////Appelle Fonction
void reglage_pins(int pin1, int pin2, int pin3, int pin4);
void prochainStep(int numero);
void marche_avant(float attente, int nombre_de_pas);
void marche_arriere(float attente, int nombre_de_pas);

void aller_retour_1(float attente);
void aller_retour_2(float attente);

void ISR_my_pc_reception();
///MAIN///
int main()
{
          my_pc.baud(115200);
    my_pc.attach(&ISR_my_pc_reception, UnbufferedSerial::RxIrq);

    servo_mot_1.period_ms(20);          // Initialisation périodel du
servomoteur 1
    servo_mot_1.pulsewidth_us(1500);

    servo_mot_2.period_ms(20);          // Initialisation période du
servomeur
    servo_mot_2.pulsewidth_us(1500);



    while(1) {
        EnaA.write(1);
        EnaB.write(1);
// 200 pas en marche avant, rotation rapide
   //     marche_avant(10000, 200);
```

```
    //      wait_us(0.01);
// 20 pas en marche arrière rotation lente
        marche_arriere(5000, 200);
        wait_us(0.05);
    }
}
```

```
// fonction tapis roulant
void reglage_pins(int pin1, int pin2, int pin3, int pin4)
{
    Bob1 = pin1;
    Bob2 = pin2;
    Bob3 = pin3;
    Bob4 = pin4;
}

void prochainStep(int numero)
{
    if (numero == 0) {
        reglage_pins(1, 0, 0, 0);
    }
    if (numero == 1) {
        reglage_pins(0, 1, 0, 0);
    }
    if (numero == 2) {
        reglage_pins(0, 0, 1, 0);
    }
    if (numero == 3) {
        reglage_pins(0, 0, 0, 1);
    }
}
void marche_avant(float attente, int nombre_de_pas)
{
    for (int i=0; i <= nombre_de_pas; i++) {
        prochainStep(i % 4);
        wait_us(attente);
    }
}

void marche_arriere(float attente, int nombre_de_pas)
{
    for (int i=0; i <= nombre_de_pas; i++) {
        prochainStep(3 - (i % 4));
        wait_us(attente);
    }
}
```

```
//Fonction servo_mot

void aller_retour_1(float attente){
        for (int i=0; i <= 1000; i++) {
        prochainStep(i % 4);
        wait_us(attente);
    }
 servo_mot_1.pulsewidth_us(2500);   // Angle négatif
         for (int i=0; i <= 500; i++) {
        prochainStep(i % 4);
```

```
        wait_us(attente);
    }
        servo_mot_1.pulsewidth_us(600);    // Angle positif
         for (int i=0; i <= 500; i++) {
        prochainStep(i % 4);
        wait_us(attente);}

}

void aller_retour_2(float attente){
        for (int i=0; i <= 1000; i++) {
        prochainStep(i % 4);
        wait_us(attente);
    }
 servo_mot_2.pulsewidth_us(2500);    // Angle négatif
         for (int i=0; i <= 500; i++) {
        prochainStep(i % 4);
        wait_us(attente);
    }
        servo_mot_2.pulsewidth_us(600);    // Angle positif
         for (int i=0; i <= 500; i++) {
        prochainStep(i % 4);
        wait_us(attente);
         }
}
```

```
//Fonction communication python-carte

void ISR_my_pc_reception(){
    my_pc.read(&data, 1);      // get the received byte
    if(data == 'a'){  aller_retour_1(5000); }

    else if(data == 'b'){aller_retour_2(5000);}
    my_pc.write(&data, 1);
}
```

Mise en commun 2.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 27 16:50:57 2023

@author: TP02
"""

#Dans cette fonction on enregistre une image et on cherche les formes dessus.
#Les formes détectables sont : rectangle, triangle, cercle, pentagone, hexagone, étoile à 6 branches. hexagone = cercle
#Libraries
from pyueye import ueye
import numpy as np
import cv2
import sys
import time
from serial import Serial
import serial.tools.list_ports

global img #variable globale

#-----------------------------------------------------------------------------------------------------
def is_SetExposureTime(hCam, EXP, newEXP):
        """
        Description

        The function is_SetExposureTime() sets the with EXP indicated exposure time in ms. Since this
        is adjustable only in multiples of the time, a line needs, the actually used time can deviate from
        the desired value.

        The actual duration adjusted after the call of this function is readout with the parameter newEXP.
        By changing the window size or the readout timing (pixel clock) the exposure time set before is changed also.
        Therefore is_SetExposureTime() must be called again thereafter.

        Exposure-time interacting functions:
            - is_SetImageSize()
            - is_SetPixelClock()
            - is_SetFrameRate() (only if the new image time will be shorter than the exposure time)

        Which minimum and maximum values are possible and the dependence of the individual
        sensors is explained in detail in the description to the uEye timing.

        Depending on the time of the change of the exposure time this affects only with the recording of
        the next image.

        :param hCam: c_uint (aka c-type: HIDS)
        :param EXP: c_double (aka c-type: DOUBLE) - New desired exposure-time.
        :param newEXP: c_double (aka c-type: double *) - Actual exposure time.
        :returns: IS_SUCCESS, IS_NO_SUCCESS

        Notes for EXP values:

        - IS_GET_EXPOSURE_TIME Returns the actual exposure-time through parameter newEXP.
        - If EXP = 0.0 is passed, an exposure time of (1/frame rate) is used.
        - IS_GET_DEFAULT_EXPOSURE Returns the default exposure time newEXP Actual exposure time
        - IS_SET_ENABLE_AUTO_SHUTTER : activates the AutoExposure functionality.
```

```
055|           Setting a value will deactivate the functionality.
056|           (see also 4.86 is_SetAutoParameter).
057|         """
058|         _hCam = ueye._value_cast(hCam, ueye.ctypes.c_uint)
059|         _EXP = ueye._value_cast(EXP, ueye.ctypes.c_double)
060|         ret = IdsCamera._is_SetExposureTime(_hCam, _EXP,
ueye.ctypes.byref(newEXP) if newEXP is not None else None)
061|         return ret
062| def set_camera_exposure(self, level_us):
063|         """
064|         :param level_us: exposure level in micro-seconds, or zero for auto
exposure
065|
066|         note that you can never exceed 1000000/fps, but it is possible to change
the fps
067|         """
068|         p1 = ueye.DOUBLE()
069|         if level_us == 0:
070|             rc = IdsCamera._is_SetExposureTime(self.hCam,
ueye.IS_SET_ENABLE_AUTO_SHUTTER, p1)
071|             print(f'set_camera_exposure: set to auto')
072|         else:
073|             ms = ueye.DOUBLE(level_us / 1000)
074|             rc = IdsCamera._is_SetExposureTime(self.hCam, ms, p1)
075|             print(f'set_camera_exposure: requested {ms.value}, got {p1.value}')
076|
077| #Variables
078| hCam = ueye.HIDS(0)             #0: first available camera;  1-254: The camera
with the specified camera ID
079| sInfo = ueye.SENSORINFO()
080| cInfo = ueye.CAMINFO()
081| pcImageMemory = ueye.c_mem_p()
082| MemID = ueye.int()
083| rectAOI = ueye.IS_RECT()
084| pitch = ueye.INT()
085| nBitsPerPixel = ueye.INT(24)    #24: bits per pixel for color mode; take 8 bits
per pixel for monochrome
086| channels = 3                    #3: channels for color mode(RGB); take 1 channel
for monochrome
087| m_nColorMode = ueye.INT()     # Y8/RGB16/RGB24/REG32
088| bytes_per_pixel = int(nBitsPerPixel / 8)
089|
#-------------------------------------------------------------------------------------
-------------------------------------------------
090| print("START")
091| print()
092| print(m_nColorMode)
093|
094| #/!\ Il faut bien fermer la camera car sinon il faut redemarrer l'ordinateur
095|
096| #Initialisation : connexion à la caméra
097| # Starts the driver and establishes the connection to the camera
098| nRet = ueye.is_InitCamera(hCam, None)
099| if nRet != ueye.IS_SUCCESS:
100|     print("is_InitCamera ERROR")
101|
102| # Reads out the data hard-coded in the non-volatile camera memory and writes it
to the data structure that cInfo points to
103| #nRet = ueye.is_GetCameraInfo(hCam, cInfo)
104| #if nRet != ueye.IS_SUCCESS:
105| #    print("is_GetCameraInfo ERROR")
106|
107| # You can query additional information about the sensor type used in the camera#
108| #nRet = ueye.is_GetSensorInfo(hCam, sInfo)
109| #if nRet != ueye.IS_SUCCESS:
110| #    print("is_GetSensorInfo ERROR")
111|
112| #/!\ Quand on enlève ça "Kernel died" on va eviter de le tuer le pauvre
```

```
113| qnRet = ueye.is_ResetToDefault( hCam)
114| if nRet != ueye.IS_SUCCESS:
115|     print("is_ResetToDefault ERROR")
116|
117| # Set display mode to DIB
118| #nRet = ueye.is_SetDisplayMode(hCam, ueye.IS_SET_DM_DIB)
119|
120| # Set the right color mode
121| if int.from_bytes(sInfo.nColorMode.value, byteorder='big') ==
ueye.IS_COLORMODE_BAYER:
122|     # setup the color depth to the current windows setting
123|     ueye.is_GetColorDepth(hCam, nBitsPerPixel, m_nColorMode)
124|     bytes_per_pixel = int(nBitsPerPixel / 8)
125|     print("IS_COLORMODE_BAYER: ", )
126|     print("\tm_nColorMode: \t\t", m_nColorMode)
127|     print("\tnBitsPerPixel: \t\t", nBitsPerPixel)
128|     print("\tbytes_per_pixel: \t\t", bytes_per_pixel)
129|     print()
130|
131| elif int.from_bytes(sInfo.nColorMode.value, byteorder='big') ==
ueye.IS_COLORMODE_CBYCRY:
132|     # for color camera models use RGB32 mode
133|     m_nColorMode = ueye.IS_CM_BGRA8_PACKED
134|     nBitsPerPixel = ueye.INT(32)
135|     bytes_per_pixel = int(nBitsPerPixel / 8)
136|     print("IS_COLORMODE_CBYCRY: ", )
137|     print("\tm_nColorMode: \t\t", m_nColorMode)
138|     print("\tnBitsPerPixel: \t\t", nBitsPerPixel)
139|     print("\tbytes_per_pixel: \t\t", bytes_per_pixel)
140|     print()
141|
142| elif int.from_bytes(sInfo.nColorMode.value, byteorder='big') ==
ueye.IS_COLORMODE_MONOCHROME:
143|     # for color camera models use RGB32 mode
144|     m_nColorMode = ueye.IS_CM_MONO8
145|     nBitsPerPixel = ueye.INT(8)
146|     bytes_per_pixel = int(nBitsPerPixel / 8)
147|     print("IS_COLORMODE_MONOCHROME: ", )
148|     print("\tm_nColorMode: \t\t", m_nColorMode)
149|     print("\tnBitsPerPixel: \t\t", nBitsPerPixel)
150|     print("\tbytes_per_pixel: \t\t", bytes_per_pixel)
151|     print()
152|
153| else:
154|     # for monochrome camera models use Y8 mode
155|     m_nColorMode = ueye.IS_CM_MONO8
156|     nBitsPerPixel = ueye.INT(8)
157|     bytes_per_pixel = int(nBitsPerPixel / 8)
158|     print("else")
159|
160| # Can be used to set the size and position of an "area of interest"(AOI) within
an image réglage de sorte à voir que le tapis
161| nRet = ueye.is_AOI(hCam, ueye.IS_AOI_IMAGE_GET_AOI, rectAOI,
ueye.sizeof(rectAOI))
162| if nRet != ueye.IS_SUCCESS:
163|     print("is_AOI ERROR")
164|
165| width = rectAOI.s32Width
166| height = rectAOI.s32Height
167|
168| # Prints out some information about the camera and the sensor
169| print("Camera model:\t\t", sInfo.strSensorName.decode('utf-8'))
170| print("Camera serial no.:\t", cInfo.SerNo.decode('utf-8'))
171| print("Maximum image width:\t", width)
172| print("Maximum image height:\t", height)
173| print()
174|
175|
```

```
#-----------------------------------------------------------------------------------------------
176|
177| # Allocates an image memory for an image having its dimensions defined by width
and height and its color depth defined by nBitsPerPixel
178| nRet = ueye.is_AllocImageMem(hCam, width, height, nBitsPerPixel, pcImageMemory,
MemID)
179| if nRet != ueye.IS_SUCCESS:
180|     print("is_AllocImageMem ERROR")
181| else:
182|     # Makes the specified image memory the active memory
183|     nRet = ueye.is_SetImageMem(hCam, pcImageMemory, MemID)
184|     if nRet != ueye.IS_SUCCESS:
185|         print("is_SetImageMem ERROR")
186|     else:
187|         # Set the desired color mode
188|         nRet = ueye.is_SetColorMode(hCam, m_nColorMode)
189|
190|
191|
192| # Activates the camera's live video mode (free run mode)
193| nRet = ueye.is_CaptureVideo(hCam, ueye.IS_DONT_WAIT)
194| if nRet != ueye.IS_SUCCESS:
195|     print("is_CaptureVideo ERROR")
196|
197| # Enables the queue mode for existing image memory sequences
198| nRet = ueye.is_InquireImageMem(hCam, pcImageMemory, MemID, width, height,
nBitsPerPixel, pitch)
199| if nRet != ueye.IS_SUCCESS:
200|     print("is_InquireImageMem ERROR")
201| else:
202|     print("Press q to leave the programm")
203|
204|
#-----------------------------------------------------------------------------------------------
205|
206| ports = serial.tools.list_ports.comports()
207| for port, desc, hwid in sorted(ports):
208|     print(port, desc)
209| selectPort = input("Select a COM port : ")
210| serNuc = Serial('COM'+str(selectPort), 115200)
211|
212| # Continuous image display
213| while(nRet == ueye.IS_SUCCESS):
214|
215|     # In order to display the image in an OpenCV window we need to...
216|     # ...extract the data of our image memory
217|     array = ueye.get_data(pcImageMemory, width, height, nBitsPerPixel, pitch,
copy=False)
218|
219|     # bytes_per_pixel = int(nBitsPerPixel / 8)
220|
221|     # ...reshape it in an numpy array...
222|     frame = np.reshape(array,(height.value, width.value, bytes_per_pixel))
223|
224|     # ...resize the image by a half
225|     frame = cv2.resize(frame,(0,0),fx=0.5, fy=0.5)
226|
227|
#-----------------------------------------------------------------------------------------------
228|     #Include image data processing here
229|
230|
#-----------------------------------------------------------------------------------------------
231|
```

```
232|     ES=cv2.getStructuringElement(cv2.MORPH_RECT,(2,2))
233|     #...and finally display it
234|     #cv2.imshow("SimpleLive_Python_uEye_OpenCV", frame)
235|     cv2.imwrite("image.jpg",frame) #Télecharge une image sur l'ordi qu'on va
exploiter
236|     img = cv2.imread("image.jpg") #Il lit l'image qu'il va enregistrer
237|     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
238|
239|     #blur = cv2.GaussianBlur(gray,(5,5),0)
240|     #ret1,thresh1 = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
241|     ret1,thresh1 = cv2.threshold(gray,248,255,cv2.THRESH_BINARY) #Seuilage qui
permet de repérer seulement la forme
242|     thresh2=cv2.erode(thresh1,ES)
243|     thresh=cv2.dilate(thresh2,ES)
244|     #cv2.imshow("tentative",thresh)
245|     contours,hierarchy = cv2.findContours(thresh, 1, 2)
246|     print("Number of contours detected:", len(contours))
247|
248|     letter="ras"
249|     if len(contours)!=0:
250|         gd_cnt=contours[0]
251|         for cnt in contours:
252|             l=len(cnt)
253|             lgd_cnt=len(gd_cnt)
254|             if l>lgd_cnt:
255|                 gd_cnt=cnt
256|
257|         cnt=gd_cnt
258|         x1,y1 = cnt[0][0]
259|         coordinates = []
260|         approx = cv2.approxPolyDP(cnt, 0.01*cv2.arcLength(cnt, True), True)
261|         if len(approx) == 4:
262|             x, y, w, h = cv2.boundingRect(cnt)
263|             ratio = float(w)/h
264|             if ratio >= 0.9 and ratio <= 1.1:
265|                 # img = cv2.drawContours(img, [cnt], -1, (0,255,255), 3)
266|                 # cv2.putText(img, 'Square', (x1, y1), cv2.FONT_HERSHEY_SIMPLEX,
0.6, (255, 255, 0), 2)
267|                 letter="S"
268|             else:
269|                 #cv2.putText(img, 'Rectangle', (x1, y1),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
270|                 #img = cv2.drawContours(img, [cnt], -1, (0,255,0), 3)
271|                 letter="R"
272|         elif len(approx)==3:
273|            coordinates.append([cnt])
274|            #cv2.drawContours(img, [cnt], 0, (0, 0, 255), 3)
275|            #cv2.putText(img, 'Triangle', (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(0, 255, 0), 2)
276|            letter="T"
277|         elif len(approx)==5:
278|            coordinates.append([cnt])
279|            #cv2.drawContours(img, [cnt], 0, (0, 0, 255), 3)
280|            #cv2.putText(img, 'Pentagone', (x1, y1), cv2.FONT_HERSHEY_SIMPLEX,
0.6, (0, 255, 0), 2)
281|            letter="P"
282|         elif len(approx)==12:
283|            coordinates.append([cnt])
284|            #cv2.drawContours(img, [cnt], 0, (0, 0, 255), 3)
285|            #cv2.putText(img, 'Etoile', (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(0, 255, 0), 2)
286|            letter="E"
287|         else:
288|            coordinates.append([cnt])
289|            #cv2.drawContours(img, [cnt], 0, (0, 0, 255), 3)
290|            #cv2.putText(img, 'Circle', (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(0, 255, 0), 2)
291|            letter="C"
```

```
292|
293|
294|
295|         #cv2.imshow("Shapes", img)
296|         print(letter)
297|         if letter=="S" :
298|             data_to_send =letter
299|             serNuc.write(bytes(data_to_send,'utf-8'))
300|             while serNuc.inWaiting() == 0:
301|                     pass
302|             data_rec = serNuc.read(1)  # bytes
303|             print(str(data_rec))
304|         time.sleep(1)
305|
306|         # Press q if you want to end the loop
307|     if letter=="E":
308|         break
309|
310| serNuc.close()
311|
#-------------------------------------------------------------------------------------
-------------------------------------------
312| #Ce qui permet de fermer la caméra
313|
314| # Releases an image memory that was allocated using is_AllocImageMem() and
removes it from the driver management
315| ueye.is_FreeImageMem(hCam, pcImageMemory, MemID)
316|
317| # Disables the hCam camera handle and releases the data structures and memory
areas taken up by the uEye camera
318| ueye.is_ExitCamera(hCam)
319|
320| cv2.waitKey(0)
321| cv2.destroyAllWindows()
322|
323|
324| # Destroys the OpenCv windows
325| cv2.destroyAllWindows()
326|
327| print()
328| print("END")
329|
```

code interface.py

```
001| # -*- coding: utf-8 -*-
002| """
003| Created on Mon Feb 27 16:44:13 2023
004|
005| @author: clara.soncin
006| """
007|
008| import sys
009|
010| from PyQt5.QtWidgets import QApplication, QDialog, QMainWindow, QPushButton,
QCheckBox
011| from PyQt5 import uic
012| from serial import Serial
013| import serial.tools.list_ports
014|
015|
016| class UI(QMainWindow): # charge la fenêtre principal de l'interface
017|     def __init__(self):
018|         super(UI,self).__init__()
019|
020|         #load the ui file : charge le code de l'interface créée
021|         uic.loadUi("fenetre1.ui", self)
022|
023|         #Define our Widgets : attribue un nom dans ce programme Python à l'action
de chaque boutton
024|         self.tri_forme = self.findChild(QPushButton,"pushButton_2")
025|         self.tri_couleur = self.findChild(QPushButton,"pushButton_3")
026|         self.arret = self.findChild(QPushButton,"pushButton_4")
027|         self.init_connex = self.findChild(QPushButton, "pushButton")
028|         self.arret_connex = self.findChild(QPushButton, "pushButton_5")
029|
030|         # Do something : attribue une fonction à une action réalisée sur un
bouton
031|         self.tri_forme.clicked.connect(self.clicker_forme)
032|         self.tri_couleur.clicked.connect(self.clicker_couleur)
033|         self.arret.clicked.connect(self.clicker_arret)
034|         self.init_connex.clicked.connect(self.clicker_init_connex)
035|         self.arret_connex.clicked.connect(self.clicker_arret_connex)
036|
037|         # Show the App : permet l'affichage de l'application
038|         self.show()
039|
040|     def clicker_forme(self): # fonction associée au bouton "Trier les formes"
041|         print("les formes fonctionnent")
042|         dialog = choix_formeDialog(self)
043|         dialog.exec()
044|
045|     def clicker_couleur(self): # fonction associée au bouton "Trier les couleurs"
046|         print("les couleurs fonctionnent")
047|
048|     def clicker_arret(self): # fonction associée au bouton "Arreter le tri"
049|         test_arret(self)
050|
051|     def clicker_init_connex(self): # fonction azsociée au bouton "initialiser la
connexion"
052|         ports = serial.tools.list_ports.comports()  #nom du port
053|
054|               # To obtain the list of the communication ports
055|         for port, desc, hwid in sorted(ports):
056|             print(port, desc)
057|               # To select the port to use
058|         selectPort = input("Select a COM port : ") #l'utilisateur rentre le
numéro du port à utiliser
059|         print("Port Selected : COM{selectPort}")
060|               # To open the serial communication at a specific baudrate
061|         serNuc = Serial('COM'+str(selectPort), 115200)  # Under Windows only, Dis
```

```
sur quel port ça écrit
062|         appOk = 1 #idée : on fait une boucle et quand on envoie q, on en sort,
sinon ça fonctionne h24
063|         while appOk:
064|             data_to_send = input("Char to send : ")
065|             if data_to_send == 'q' or data_to_send == 'Q':
066|                 appOk = 0
067|             else:
068|                 serNuc.write(bytes(data_to_send, "utf-8")) #écrit sur la carte le
caractère reçu
069|             while serNuc.inWaiting() == 0:
070|                 pass
071|             data_rec = serNuc.read(1)  # bytes, confirmation de l'envoie
072|             print(str(data_rec))
073|
074|         serNuc.close() #on sort du port utilisé
075|
076|     def clicker_arret_connex(self): # fonction associée au bouton "arreter la
connexion"
077|         print("arret ok")
078|
079|
080|
081| class choix_formeDialog(QDialog): # charge la fenêtre secondaire de l'interface
associée au bouton "Trier les formes"
082|     def __init__(self, parent=None):
083|         super().__init__(parent)
084|
085|         #load the ui file
086|         uic.loadUi("choix_formes.ui", self)
087|
088|         #Define our Widgets : association des boutons à un nom sous Python
089|         self.carre = self.findChild(QCheckBox,"checkBox")
090|         self.rectangle = self.findChild(QCheckBox,"checkBox_2")
091|         self.triangle = self.findChild(QCheckBox,"checkBox_3")
092|         self.etoile = self.findChild(QCheckBox,"checkBox_3")
093|         self.cercle = self.findChild(QCheckBox,"checkBox_3")
094|         self.autre = self.findChild(QCheckBox,"checkBox_3")
095|
096|         # Do something
097|
098|         # Show the App
099|         self.show()
100|
101|
102| def test_arret(self): # fonction externe associée au bouton "arreter le tri" sert
de test pour utiliser des fonctions externe
103|     print("le test arret fonctionne")
104| #initialize the App
105| app = QApplication(sys.argv)
106| UIWindow = UI()
107| app.exec_()
```