Bras articulé robotisé, optimisant l'éclairage rasant en vision industrielle

Rapport technique

COLLIGNON Martin
DE SALEON Victoire
LAPRAS Albane
ZHU Tong

ProTIS - Procédés de Traitement de l'Information et du Signal
Institut d'Optique / 2A

Date de rédaction : 29/03/22 – 08/04/22

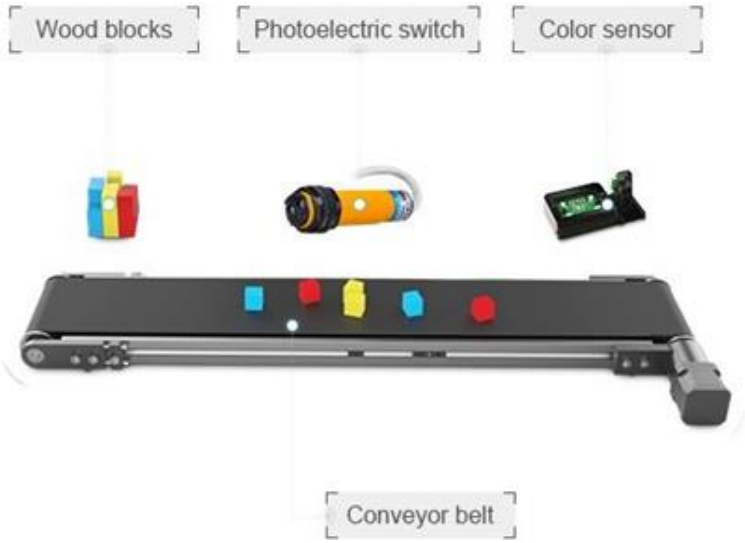
Sommaire

1. Introduction

1.1 Descriptif du projet

Le contrôle non destructif (CND) de la qualité des objets sur une chaîne de production est fondamental pour pouvoir exclure les produits défectueux ou stopper la production dans le cas d'un défaut systématique. Une solution de CND permettant la production de masse haute cadence tout en assurant la qualité de fabrication des produits est la vision industrielle.

Pour répondre aux besoins d'industriels souhaitant automatiser leur chaîne de production en ajoutant des options de tri de pièces ou d'objets, la société SOLEC souhaite proposer une solution de tri d'objets en fonction de leur couleur et/ou leur forme. Ce système devrait se baser sur un convoyeur Dobot entraîné par un moteur pas-à-pas, un détecteur de couleur et un détecteur de présence de pièces.



Source : Convoyeur DOBOT (Offre de projet – Automatisation d'une chaine de tri)

Un projet « PIMS Vision industrielle » a donc vu le jour en octobre 2021 afin de développer un dispositif capable de détecter les caractéristiques des objets qui défilent sur un convoyeur roulant.

(**Figure 1**)

Ce démonstrateur doit réaliser une acquisition optimale d'images à traiter (éclairage et vitesse de défilement adaptés) et offrir une interface graphique permettant de piloter le système et de traiter les images acquises par la caméra uEye pour en extraire les caractéristiques (identification de formes, de contours, couleurs …). En janvier 2022, la solution développée par l'équipe « PIMS Vision industrielle » utilise un éclairage surplombant diffus et se heurte alors à une difficulté de détection de détails ou de défauts de faibles reliefs sur certains objets (comme les pièces de monnaie). Ces détails ne sont détectables qu'avec un éclairage rasant.

Dans le but de perfectionner cette solution en partenariat avec l'équipe « PIMS Vision industrielle », notre équipe ProTIS a développé, de janvier à avril 2022, un bras articulé automatisé qui permet de positionner un éclairage rasant autour des pièces présentant des gravures, et respectant la cadence du convoyeur.

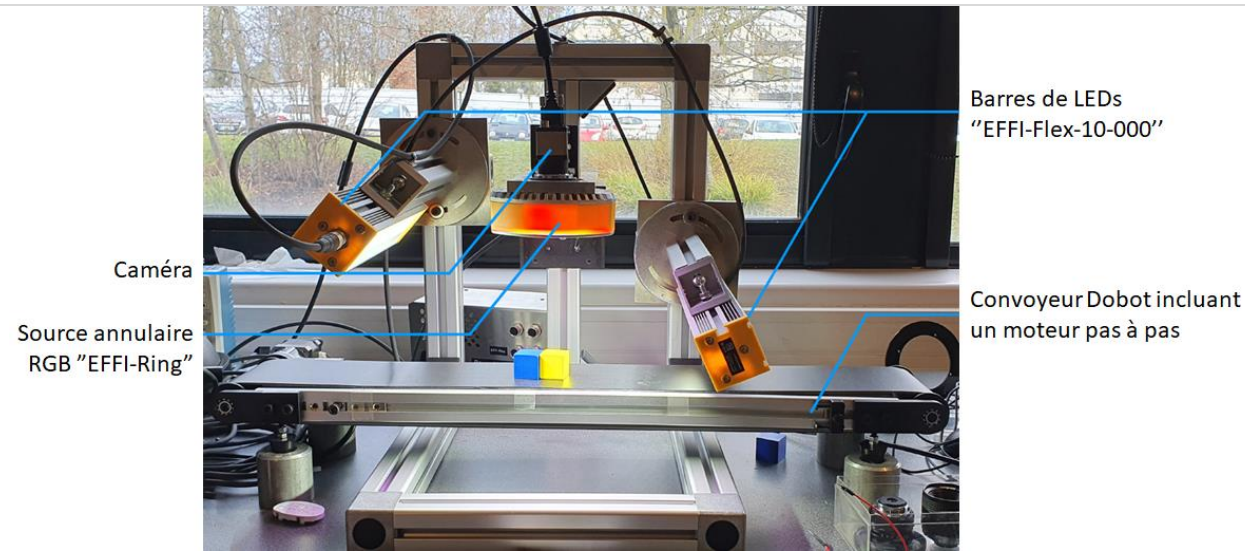


***Figure 1** : Photo du montage avec le convoyeur, la caméra et des sources d'éclairages disponibles (PIMS) sans l'éclairage rasant*

1.2 Cahier des charges

Le bras articulé robotisé, comportant sur son extrémité un anneau de LEDs rasant rasant EFFI-LLA (EFFI-Lux) monochrome rouge (**Figure 2**), doit réaliser les fonctions suivantes :

- Sortir de sa position de repos (bras replié en dehors du convoyeur) et effectuer une rotation à +90° afin de positionner l'anneau de LEDs en dessous de la caméra et au-dessus de l'objet sur le convoyeur : **« déploiement du dispositif »**
- Descendre (translation verticale) au niveau du convoyeur pour permettre l'éclairage rasant et la prise d'une photo avec la caméra sans arrêter le convoyeur. La photo doit s'enregistrer automatiquement dans un dossier de l'ordinateur.
- Remonter (translation verticale)
- Remonter et redescendre autant de fois que nécessaire, pour le contrôle d'autres objets arrivant sur le convoyeur en fonctionnement, jusqu'au rangement du dispositif
- Revenir à sa position de repos initiale en dehors du convoyeur (rotation de -90°), après avoir éteint l'anneau de LEDs : **« rangement du dispositif »**

***Figure 2** : Anneau de LEDs rasant EFFI-LLA (Source : EFFI-Lux)*

La commande du bras articulé doit pouvoir se faire de deux façons différentes pour l'utilisateur :

- soit par un écran tactile qui permet d'allumer et éteindre l'éclairage de l'anneau mais aussi de de contrôler les différents éclairages du dispositif et de régler la vitesse et le sens de défilement du convoyeur. Cet écran tactile permet un **pilotage manuel du bras.**
- soit par une interface sur ordinateur qui permet également d'allumer et éteindre l'éclairage de l'anneau, de contrôler les différents éclairages du dispositif, de régler la vitesse et le sens de défilement du convoyeur et mais aussi d'effectuer un traitement (en temps réel) des images acquises par la caméra pour les détails ou les défauts de faibles reliefs de certains objets (éclairage rasant). L'interface sur ordinateur permettra un **pilotage manuel du bras ou fonctionnement automatisé du dispositif de vision industrielle en éclairage rasant.**

Ces commandes doivent donc s'intégrer dans le programme existant de PIMS en rajoutant de nouvelles fenêtres dans l'interface existante pour y inclure le pilotage du bras et le paramétrage du traitement d'images avec un éclairage rasant. La structure générale de l'interface « PIMS vision industrielle » est donnée en partie 3.2 (Conception – partie informatique).

D'autres contraintes et performances sont aussi à prendre en compte :

- La principale difficulté de ce contrôle par vision industrielle est que la caméra est fixe et que les objets sont en mouvement sur le convoyeur pendant le temps de l'acquisition et de traitement des images (afin de ne pas ralentir la production). La cadence maximale du convoyeur fonctionnement (environ 55 mm/s) est donc à respecter.
- L'utilisateur aura la possibilité de choisir une hauteur de placement de l'anneau, permettant l'éclairage rasant, en respectant les dimensions des objets et du convoyeur.
- Lorsque l'anneau arrive au niveau du convoyeur (pour permettre l'éclairage rasant et optimiser l'acquisition des images et le traitement d'images associé), l'objet doit se trouver centré par rapport à l'anneau.
- L'éclairage directionnel de l'anneau de LEDs rasant EFFI-LLA (EFFI-Lux), qui s'effectue en champ sombre, peut subir des réflexions spéculaires pour les objets réfléchissants et peut donner lieu à une saturation de l'image et à l'apparition d'ombre lors du traitement des images.

2. Découpage fonctionnel

Pour répondre au cahier des charges, nous avons donc élaboré le découpage fonctionnel ci-dessous (**Figure 3**)

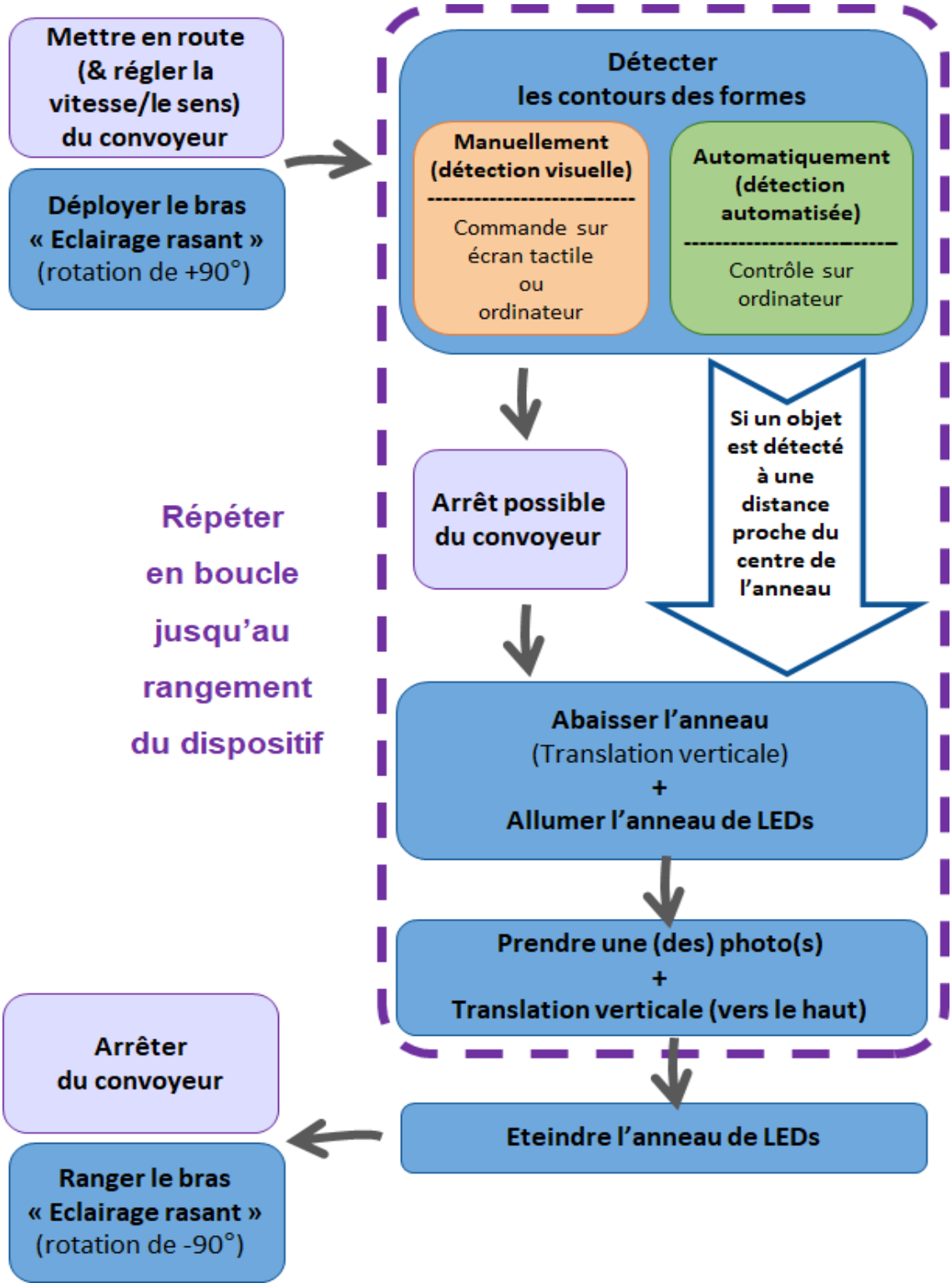
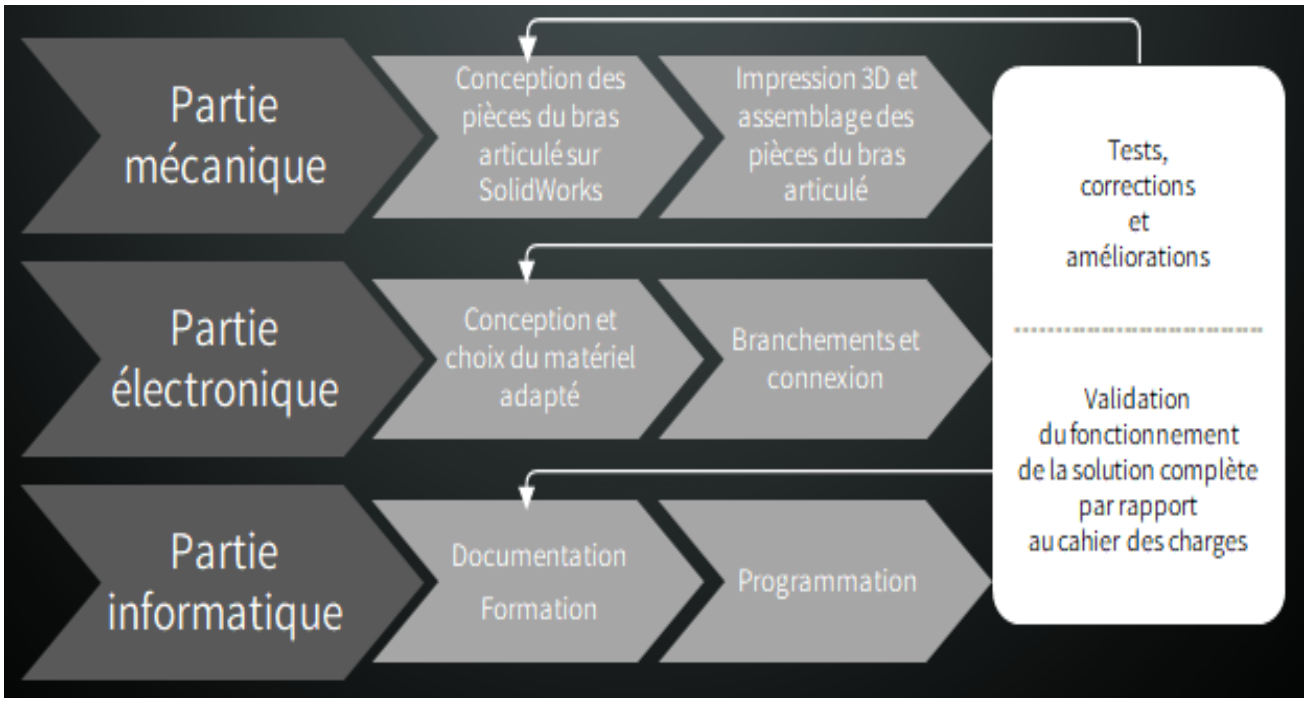


Figure 3 : *Découpage fonctionnel du dispositif*

3. Conception et réalisation

La conception et réalisation de notre solution s'est développé autour de trois parties : une partie mécanique, une partie électronique et une partie informatique.



3.1 Partie mécanique

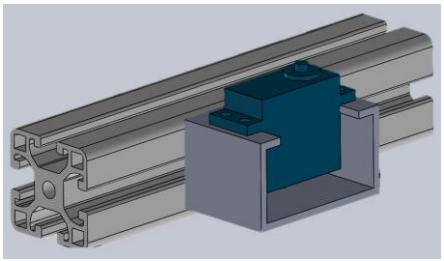
Dans la partie mécanique, nous avons conçu les différentes pièces du bras articulé sur Solidworks que nous avons ensuite imprimées en 3D. Puis nous les avons testées après assemblage d'un premier 1er bras articulé, ce qui nous a permis d'apporter les corrections et améliorations nécessaires pour aboutir à un 2ème bras plus performant.

3.1.1. Conception des pièces du bras articulé sur Solidworks

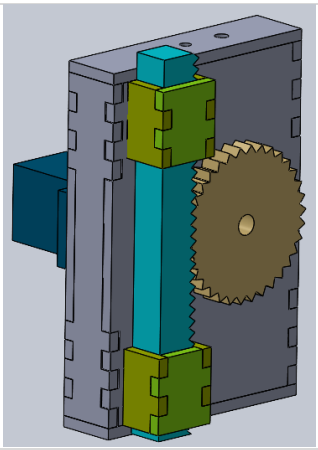
Le bras possède 2 degrés de liberté (mouvement de translation et mouvement de rotation) et se décompose en 3 parties :

- ✓ La base (verticale), fixée au support, dans laquelle est situé le 1er moteur qui fait tourner le reste du bras autour d'un axe vertical (**image a. de la Figure 4**)
- ✓ La 2e partie (verticale) au-dessus comporte un 2e moteur qui fait translater la 3e partie de façon verticale grâce à une crémaillère et un pignon (**image b. de la Figure 4**). Cela permet de convertir un mouvement de rotation en un mouvement de translation.
- ✓ La 3e partie (horizontale) à laquelle est fixée l'anneau d'éclairage (**image c. de la Figure 4**). Pour tenir l'anneau, l'extrémité du bras a une forme arrondie (même forme que l'anneau). L'anneau est fixé directement au bras à l'aide de vis.

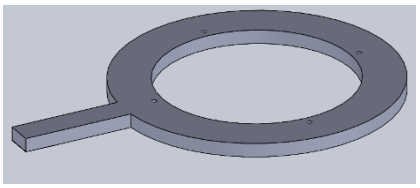
Nous avons tout d'abord conçu différentes pièces du bras articulé sur Solidworks : pièce du bas, pièce du dessus, pièce qui tient l'éclairage puis nous les avons assemblées pour former le montage présent en **annexe 1**.



a. Base avec le servomoteur qui fait tourner le reste du bras



b. Partie avec le deuxième servomoteur permettant de translater le support de l'anneau



c. Support de l'anneau avec des trous de vis pour fixer l'éclairage

***Figure 4** : Différentes parties du bras réalisées avec Solidworks*

3.1.2. Principaux verrous technologiques levés

Nous avons réalisé une première version du bras (**Figure 5**) qui présentait un certain nombre de problèmes :

- Le servomoteur du bas qui permet la rotation du bras patine de temps en temps et n'entraîne pas correctement le bras en rotation, ce qui est dû à une mauvaise fixation du servomoteur.
- Les frottements de la fixation métallique de la partie supérieure gênent la rotation du bras.
- Le bras ne peut pas faire exactement une rotation de 90° car il est fixé à une barre qui l'empêche de se replier complètement. De plus, le bras n'arrive pas parfaitement en-dessous de la caméra lorsqu'il est perpendiculaire au convoyeur.
- La partie horizontale du bras ploie légèrement en avant à cause du poids de l'anneau.
- Le centre de l'anneau de LEDs n'est pas confondu avec le centre de la caméra car le bras est visiblement trop long.

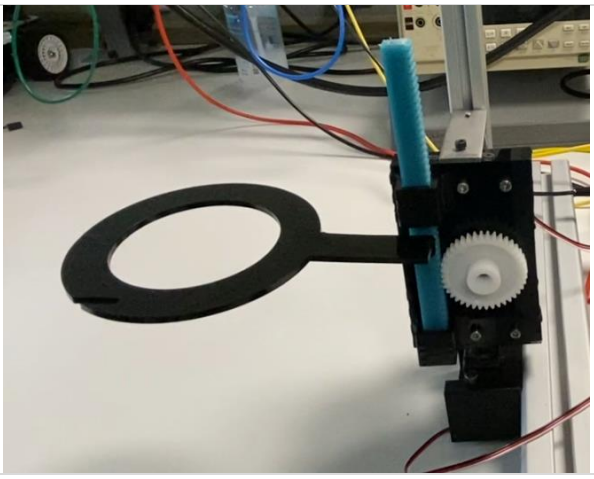


***Figure 5** : Photo du premier bras réalisé*

Nous avons donc réalisé un 2ème bras qui règle ces problèmes. (**Figure 6)**

- Nous avons tout d'abord amélioré la fixation du servomoteur du bas au bras ce qui a réglé les soucis de patinage.
- Nous avons également modifié la fixation du bras pour qu'il puisse faire exactement une rotation de 90° et qu'il arrive parfaitement en-dessous de la caméra lorsqu'il est perpendiculaire au convoyeur. Ceci a également permis de régler les problèmes de frottements de la fixation métallique de la partie supérieure qui gênaient la rotation du bras.
- Nous avons enfin réduit la longueur de la partie horizontale du bras pour éviter qu'il ne ploie trop en avant à cause du poids de l'anneau de LEDs. Ceci a également permis de superposer le centre de l'anneau de LEDs avec le centre de la caméra.

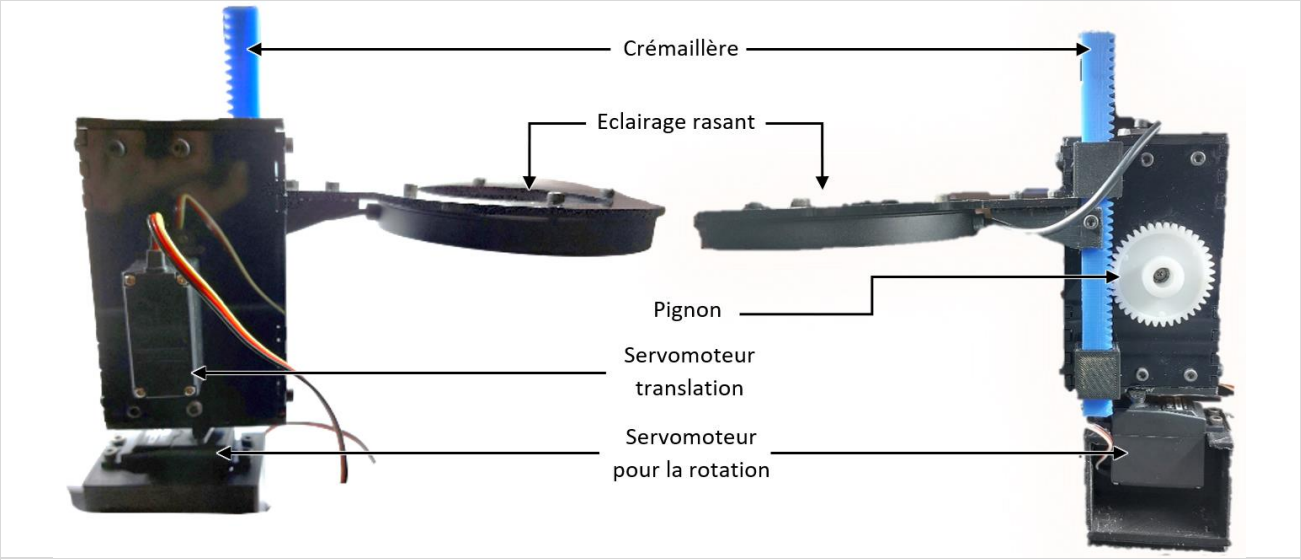


***Figure 6** : Photos du deuxième bras réalisé*

3.2 Partie électronique

3.2.1. Conception et réalisation du montage de la partie électronique

Notre dispositif est commandé par deux systèmes :

- Un écran tactile, programmé par un Code en C avec l'outil de développement Mbed, à partir d'une carte électronique (**Disco-F746NG**) qui permet à la fois de gérer l'éclairage de l'anneau de LEDs (allumer ou éteindre), de déployer ou ranger le bras, de monter ou descendre verticalement le bras au niveau du convoyeur (position du curseur pour le contrôle) et de contrôler le convoyeur (marche/arrêt, vitesse et sens).
- Une interface sur ordinateur qui permet non seulement d'effectuer les mêmes pilotages que sur l'écran tactile mais aussi de réaliser un traitement des images acquises par la caméra en éclairage rasant. Cette interface est programmée grâce à un code en Python avec la bibliothèque OpenCV pour le traitement d'image et QT l'interface de programmation d'application orientée objet.

L'équipe PIMS a réalisé un boîtier permettant de simplifier l'utilisation du montage avec l'écran tactile de la carte **Disco**. Il contient en plus de la carte, une alimentation reliée au réseau électrique du bâtiment qui permet d'alimenter les sources d'éclairage et le convoyeur. Il dispose également de ports permettant de connecter les éclairages ainsi qu'un port USB pour pouvoir connecter la carte à un ordinateur. Cette connexion USB permet d'alimenter la carte **Disco** mais aussi d'envoyer des caractères de l'ordinateur à la carte lorsque l'utilisateur a choisi le mode Ordinateur.

Nous avons choisi d'utiliser deux servomoteurs plutôt que des moteurs pas à pas d'une part parce que nous n'avons pas besoin d'un grand déplacement pour la translation du bras et d'autre part pour leur simplicité de mise en œuvre.

Dans un premier temps, avec le 1er bras, la commande des servomoteurs s'est faite à l'aide d'une deuxième carte branchée à une alimentation 5V externe. Nous avons utilisé pour cela une carte **Nucléo L476RG**. Cette solution était un peu encombrante et peu pratique donc nous avons changé de méthode pour le 2ème bras.

Dans un deuxième temps, avec le 2ème bras, nous avons souhaité implémenter le montage du bras à celui du montage global du PIMS. Pour cela, nous avons soudé les fils correspondant à l'alimentation des servomoteurs sur l'alimentation du boîtier. Nous avons également relié les fils de commande des servomoteurs aux broches D0 et D1 de la carte **Disco**. L'écran tactile du boîtier permet donc de contrôler, en plus des éclairages et du convoyeur, les deux servomoteurs responsables de la rotation et de la translation verticale du moteur. (**Figure 7**)

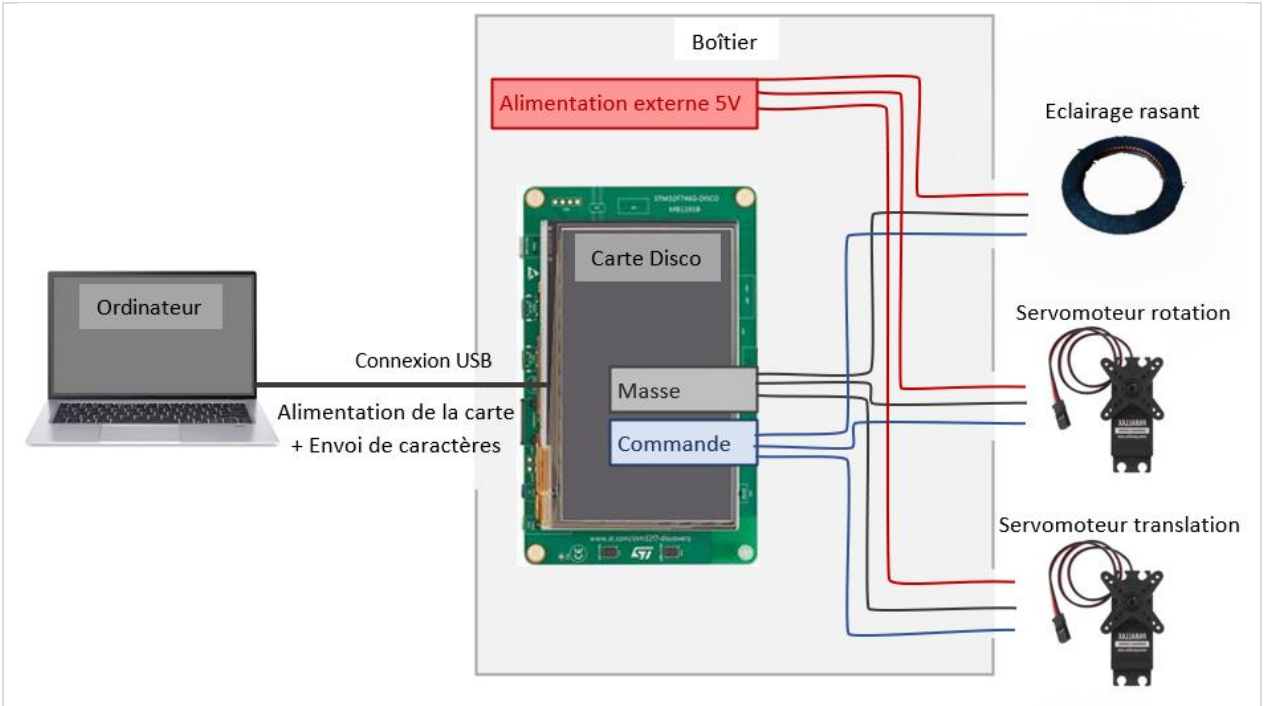


***Figure 7** : Schéma électronique simplifié du montage*

3.2.2. Calcul des commandes des servomoteurs

Un servomoteur est un actionneur dont on peut commander de façon externe la rotation calibrée. Le signal de commande est rectangulaire de période fixée à 20 ms mais la durée du temps haut est variable et c'est elle qui permet de modifier l'angle de sortie du servomoteur. Celui-ci peut effectuer des rotations comprises entre -90° et +90° par rapport à son axe principal.

Dans le cadre de notre projet, deux servomoteurs de référence **Parallax Continuous Rotation Servo (#900-00008)** sont utilisés :

- ✓ un pour déployer ou ranger le bras (rotation de 90°),
- ✓ un autre pour descendre le bras au niveau du convoyeur (translation verticale de 28 mm).

Par rapport à la datasheet, nous avons adapté les commandes spécifiées (durée du temps haut) pour que le servomoteur tourne exactement de l'angle souhaité par rapport à notre application. En effet, nous avons constaté un léger décalage entre les commandes indiquées dans la datasheet et les commandes à réaliser en pratique pour obtenir avec précision l'angle voulu. Nous avons donc déterminé les correspondances (pour le premier bras, puis pour le deuxième bras) entre la commande à appliquer (durée du temps haut) et l'angle de rotation d'un servomoteur.

- Pour le servomoteur qui permet la rotation du bras :
  Nous avons déterminé que la position angulaire de 0° correspond à un temps haut de 0,7 ms et que la position angulaire de 90° correspond à un temps haut de 1,5 ms (pour un signal de période 20 ms).
  Nous en avons déduit la loi affine : $t = 8{,}89 * \theta + 700$, avec t : la durée du temps haut (en μs) et θ : l'angle de rotation correspondant (en degré).

- Pour le servomoteur qui permet la translation verticale du bras :
  Nous avons tout d'abord calculé la correspondance entre l'angle de rotation du moteur et la distance de translation de la crémaillère : 28,5 degrés de rotation du moteur correspondent à une translation verticale du bras de 1cm.
  Ensuite, nous avons déterminé que la position verticale de 0 mm (anneau au niveau du convoyeur) correspond à une position angulaire de 0° et à un temps haut de 2,4 ms et que la position verticale de 28 mm (anneau à sa hauteur maximale) correspond à une position angulaire de 79,8° et à un temps haut de 1,6 ms.
  Nous en avons déduit la loi affine : $t = -10{,}0 * \theta + 2400$, avec t : la durée du temps haut (en μs) et θ : l'angle de rotation correspondant (en degré).

3.3 Partie informatique

Pour la partie informatique, nous avons dû partir de la solution existante « PIMS vision industrielle » qui proposait deux interfaces pour l'utilisateur : une sur l'écran tactile d'un boîtier et l'autre générée par le code Python sur l'ordinateur. Ces deux interfaces se présentent sous la forme d'une fenêtre principale qui va ensuite appeler d'autres fenêtres suivant le menu choisi.

3.3.1. Mode « écran tactile »

L'interface tactile développée par l'équipe PIMS (**Figure 8**) permet de commander manuellement :

- Les différents éclairages du dispositif : l'anneau EFFI-Ring RGB autour de la caméra (boutons RED, GREEN, BLUE), les barres de LEDs blanches EFFI-Flex W (boutons Flex1 et Flex2) et l'éclairage lambertien du dôme (bouton Dome). Chaque bouton présent sur la fenêtre principale permet d'allumer ou éteindre chacune de ses lampes indépendamment, ou de les allumer ou de les éteindre toutes ensemble (boutons ALL ON et ALL OFF),
- La mise en marche et l'arrêt (bouton On/Off), la vitesse (curseur) et le sens de défilement (bouton Direction) du convoyeur.

Nous avons donc créé dans cette interface un nouveau bouton « rasant » qui ouvre une nouvelle fenêtre (**Figure 9**) qui permet de contrôler manuellement le bras articulé équipé de son anneau de LEDs. Cette deuxième fenêtre, appelée à partir du bouton « rasant », comporte :

- Un bouton ON / OFF (rouge) qui permet de déployer le bras et d'allumer l'anneau de LEDs ou d'éteindre l'anneau et de replier le bras,
- Un bouton « Bas » qui permet de positionner l'éclairage rasant en bas (au niveau du convoyeur), un bouton « Haut » qui donne la possibilité de le positionner en haut (hauteur maximale du bras) et un curseur pour ajuster sa position verticale,
- Un bouton On/Off (gris) qui permet de mettre en marche et d'arrêter le convoyeur, un bouton Direction qui donne la possibilité de choisir le sens de défilement du convoyeur et de régler la vitesse du convoyeur à l'aide d'un curseur.

La croix en rouge (en haut à droite de l'écran) offre la possibilité de revenir sur la fenêtre générale initiale de l'écran tactile.

***Figure 8** : Fenêtre générale de l'écran tactile*

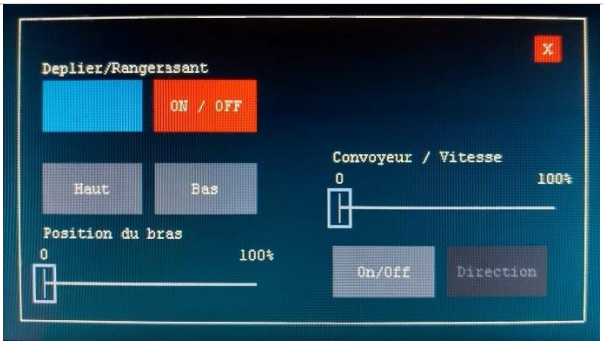


***Figure 9** : Fenêtre secondaire : menu « rasant » de l'écran tactile*

a) <u>Structure générale du code existant – mode écran tactile (PIMS)</u>

Le code présent dans la carte Disco provient de 3 fichiers codés en C sur Mbed : **`main.ccp`**, **`Vision_Indus_IHM.ccp`** et **`Vision_Indus_IHM.h`** qui sont ceux de l'**annexe 2**.
Le « `main` » commence d'abord par initialiser l'interface de l'écran tactile de la carte dans le mode Ordinateur, puis il appelle la fonction « **`chose_mode`** » qui permet de choisir le mode dans lequel l'utilisateur veut utiliser le dispositif. Cette fonction est appelée toutes les 0,01s. Pour effectuer des actions à chaque fois que cette fonction est appelée, la carte Disco reçoit des caractères depuis l'ordinateur via une connexion USB. En particulier, le caractère codé 112 en ASCII permet de se placer en mode ordinateur et le caractère codé 113 permet de se placer dans le mode écran tactile.

Le passage en mode écran tactile peut se faire de deux façons :

- ✓ soit le temps écoulé depuis le dernier caractère envoyé par l'ordinateur dépasse 5 secondes, la carte passe alors automatiquement dans le mode « écran tactile »,
- ✓ soit le mode ordinateur est utilisé et l'utilisateur clique sur le bouton « Disconnect » qui envoie le caractère 113 à la carte pour changer de mode.

Une fois le mode écran tactile enclenché, la carte utilise la fonction **`initIHM(113)`** pour initialiser l'ensemble des éléments qui composent l'interface (zones de texte, boutons, curseurs). Après initialisation, le code utilise la fonction **`updateIHM`** qui permet de mettre à jour l'interface

de l'écran tactile (couleurs, accès aux boutons) et d'effectuer des actions (allumer/éteindre un type d'éclairage, changer la vitesse du convoyeur, son sens de défilement, …) selon ce que l'utilisateur touche sur l'écran, et à l'aide d'une grande boucle **`if`**. Cette fonction est effectuée jusqu'à ce que l'utilisateur décide de changer de mode en se connectant avec l'interface de l'ordinateur qui envoie le caractère 112.

Avant d'avoir implémenté le code permettant le contrôle des 2 servomoteurs du bras, le bouton « Rasant » visible sur la **figure 8** permettait seulement d'allumer ou d'éteindre l'éclairage rasant.

b) Méthode employée pour ajouter la nouvelle fenêtre (rasant) dans la structure générale PIMS

Pour rendre clair l'utilisation du bras, pour ne pas saturer l'interface déjà existante et pour tenter de séparer les 2 projets, nous avons choisi de réaliser un deuxième menu qui s'ouvre en appuyant sur le bouton « Rasant ». Le menu qui s'ouvre alors est celui de la **figure 9**.
Pour créer cette nouvelle interface, il a fallu implémenter dans le code existant les objets graphiques (textes, boutons, curseur) dans le fichier **`Vision_Indus_IHM.ccp`**. Ces éléments sont définis entre les lignes 147 et 163, avant le corps des fonctions. (**Annexe 2**)

Comme ce menu s'ouvre lorsque l'utilisateur est en mode écran tactile, donc lorsque la fonction **`updateIHM`** est appelée en boucle, il faut initialiser la nouvelle sous-interface au sein de cette fonction (entre les lignes 382 et 406). Cette initialisation se fait une seule fois, lorsque l'utilisateur appuie sur le bouton « rasant ».
Une fois initialisée (couleurs des boutons, bras rangé, éclairage éteint), la boucle **`while(1)`** permet de rester dans cette configuration jusqu'à ce que l'utilisateur appuie sur le bouton « X » pour fermer le menu. Dans cette boucle se trouvent tous les tests permettant de vérifier si l'utilisateur appuie sur un bouton du menu. La carte réalise alors des actions en conséquence.

Pour sortir de cette sous-interface, l'utilisateur appuie donc sur « X ». La carte effectue alors différentes tâches : éteindre l'éclairage rasant, le placer au plus haut puis tourner le bras pour le ranger et enfin initialiser l'écran dans le mode écran tactile pour retrouver l'interface initiale.
La ligne 530 (**`break;`**) permet de quitter la boucle **`while(1)`**.

3.3.2. Mode « ordinateur »

L'interface Python sur l'ordinateur permet non seulement d'effectuer les mêmes commandes que sur l'écran tactile, via la communication (liaison série) avec la carte **Disco**, mais aussi d'effectuer un traitement des images acquises par la caméra en temps réel. L'ordinateur est en effet relié à la carte par voix USB et envoie des caractères qui permettent d'effectuer les différentes tâches. (**Figure 10**)

Pour se placer dans le mode ordinateur, l'utilisateur se connecte à la carte avec l'interface de l'ordinateur en envoyant le caractère 112 puis il envoie régulièrement le caractère 114 pour que la carte reste dans le mode ordinateur. Une fois dans ce mode, la carte utilise la fonction **`initIHM(112)`** pour écrire le texte « mode ordinateur » sur l'écran de la carte. Ensuite, la fonction **`computer(data_received)`** est appelée toute les 0.01s pour effectuer les actions selon le caractère reçu par la carte. Cette fonction est détaillée dans l'**annexe 2**. Elle utilise en particulier un **`switch`** qui permet de lister toutes les actions possibles. Pour régler la vitesse du convoyeur et la

translation verticale de l'éclairage rasant, on utilise des **`if`** qui permettent de calculer à l'aide de lois déterminées expérimentalement la période du signal PWM de commande pour le moteur du convoyeur, et la durée du front montant pour le signal PWM du servomoteur responsable de la translation verticale de l'éclairage.

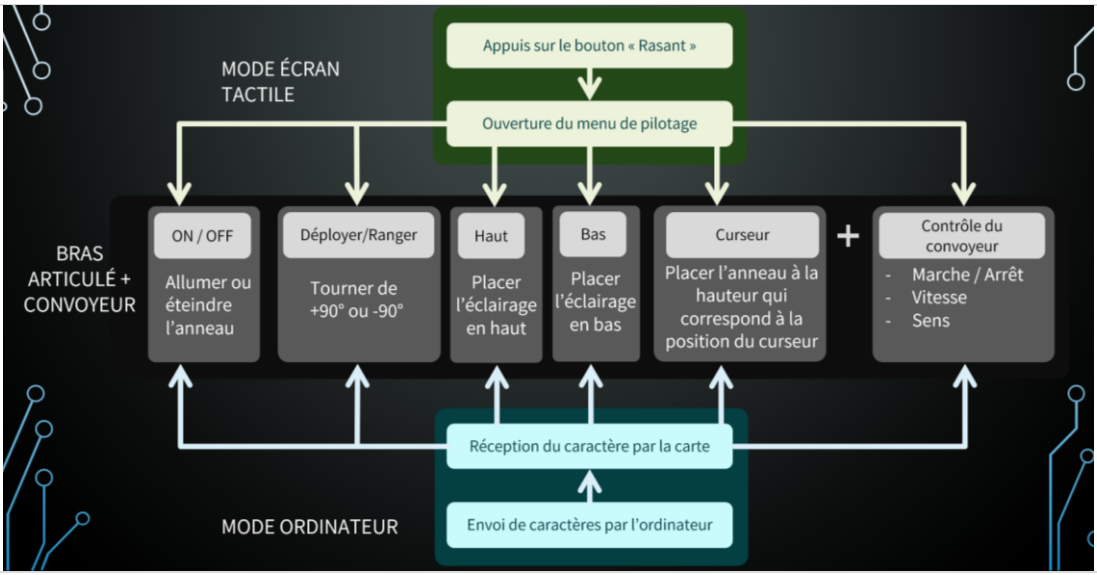


***Figure 10** : Le mode « ECRAN TACTILE » et le mode « ORNINATEUR »*

a) <u>Structure générale du code existant - mode ordinateur (PIMS)</u>

Les différents fichiers python de l'interface sont listés dans la figure ci-dessous. (**Figure 11**)
Chacune des fenêtres du programme est créée et codée sous forme de « Class ». Par exemple, dans le fichier **`main.py`**, on trouve la classe « **`ChooseCameraWindow`** » qui est constituée uniquement d'un menu déroulant dans lequel l'utilisateur peut sélectionner une caméra parmi celles qui sont branchées à son ordinateur, et d'un bouton pour valider et lancer l'application principale.
Dans la fonction d'initialisation de la classe « **`MainWindow`** », on définit les boutons du menu déroulant permettant d'ouvrir les différents menus de traitement de l'image et on les relie aux classes associées dans les fichiers pythons correspondants.

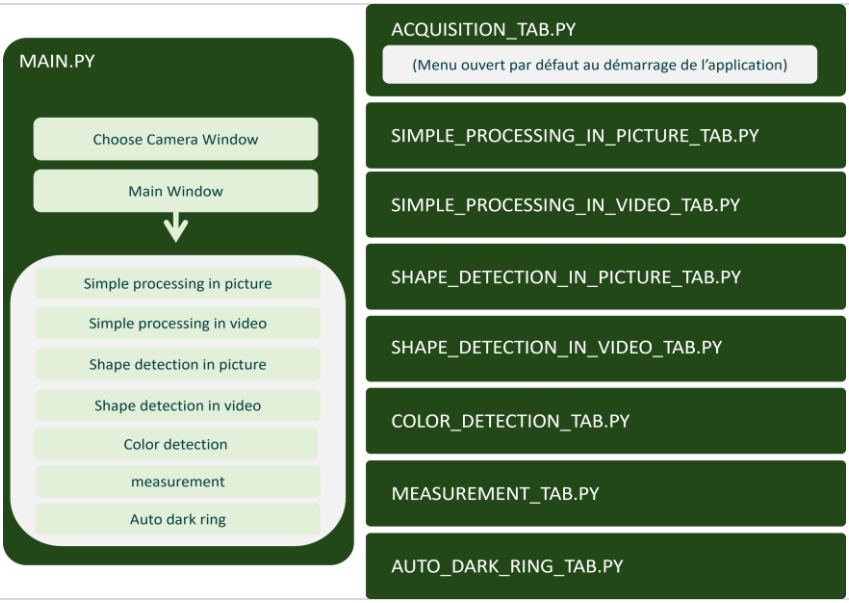


***Figure 11** : Principaux fichiers python constituant le programme de l'interface*

Voyons plus en détails comment l'option placement automatique de l'anneau a été intégrée au code.

b) Méthode employée pour ajouter la fenêtre (AutoDarkRingTab) dans la structure générale PIMS

Afin de rajouter un menu dans la fenêtre principale, la classe **`AutoDarkRingTab`** est appelée dans le **`main.py`** à la ligne 206 (dans la classe **`MainWindow`**) par :

```
@Qtc.pyqtSlot()
def add_auto_dark_ring(self):
    self.main_tab_widget.addTab(AutoDarkRingTab(self.video_thread, self.ser),
STR_AUTO_DARK_RING)
```

On aura bien entendu préalablement importé la fonction à la ligne 49

```
from AutoDarkRingTab import AutoDarkRingTab
```

Les variables passées en argument de la fonction ajoutant le tab « **`AutoDarkRingTab`** » sont donc **`self.video_thread`** (le flux d'images constituant la vidéo acquise par la caméra) et **`self.ser`** (les données liées à la communication par USB).

Le code pour l'interface sur l'ordinateur est présenté en **annexe 3**.

c) Fonctionnalités du nouveau menu

Ce menu permet tout d'abord de :
- Détecter les contours des objets,
- Déterminer leur centre et sa distance avec le centre de l'anneau,
- Baisser l'anneau (si l'objet est à moins de 250 pixels du centre de l'anneau) et prendre une photo enregistrée automatiquement dans un dossier,
- Relever l'anneau (si l'objet est à plus de 250 pixels du centre de l'anneau).

Ce menu permet également de faire du processing sur une vidéo et, en particulier, de la détection de contours lors de la vidéo, de la détection de la taille des formes et de la détection des couleurs. Plus précisément, on peut :

- Visualiser l'acquisition vidéo si la caméra est allumée (les images de la vidéo sont dans **`self.shown_cv_img`**),
- Effectuer des traitements simples de l'image (ouverture, fermeture, « flou gaussien » pour une binarisation optimale),
- Définir les paramètres de détection de contours des objets (aire minimale et maximale des objets d'intérêts),
- Enregistrer une image de l'acquisition manuellement (soit dans le clipboard soit dans un dossier).

Le processus pour lancer l'acquisition automatique des photos est le suivant :
1. Lancer le programme,
2. Sélectionner la caméra (branchée à l'ordinateur). L'utilisateur arrive alors sur la fenêtre ci-dessous (**Figure 12**). Il existe deux possibilités pour visualiser une image : soit en couleur (checkbox RGB), soit en noir et blanc (checkbox Greyscade).

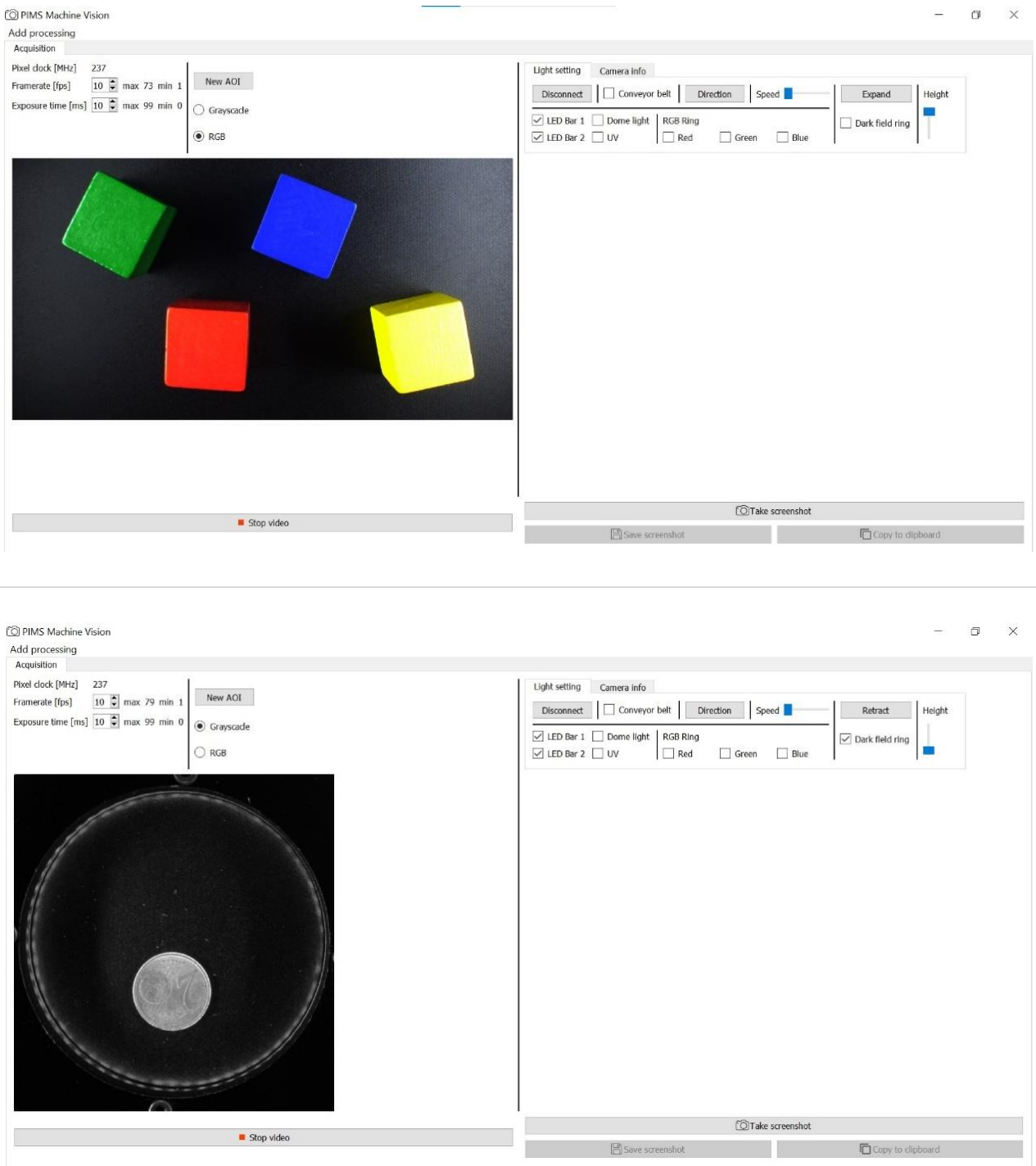


Figure 12 *: Captures d'écran de la fenêtre initiale* (**`Acquisition Tab`**)
pour visualiser une image en couleur (en haut) ou en noir et blanc (en bas)

3. Allumer et déployer le bras de l'éclairage rasant grâce aux boutons en haut à droite de la fenêtre (checkbox «**`Dark Field Ring`**», bouton « **`Expand/Retract`**», curseur « **`Height`**»),
4. Lancer l'acquisition vidéo (boutons en bas à gauche) et sélectionner la zone d'intérêt grâce au bouton **`New AOI`** (carré centré sur le centre de l'anneau),
5. Ouvrir le menu «**`AutoDarkRing`**». L'utilisateur arrive alors sur la fenêtre ci-dessous : (**Figure 13**)

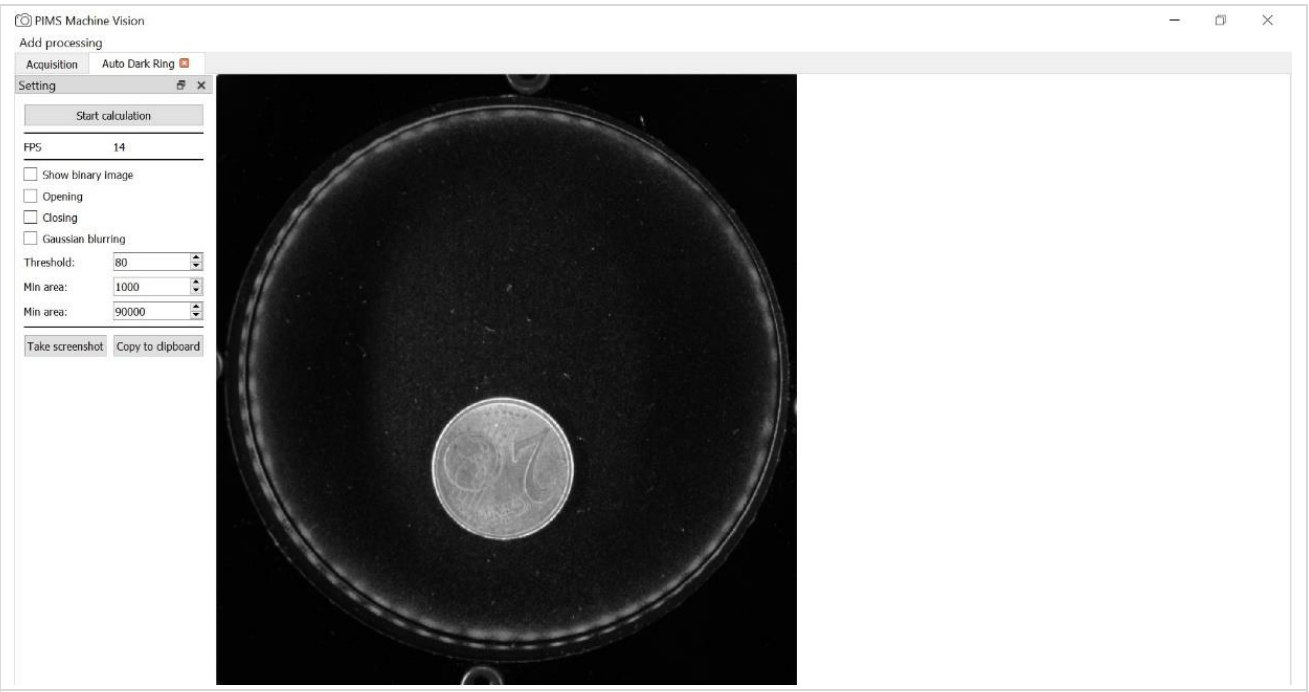


***Figure 13** : Captures d'écran de la fenêtre du bras articulé* (**`Auto Dark Ring Tab`**)

6. Cliquer sur le bouton «**`Start calculation`**».

   Ce dernier permet de :
   - Détecter les contours des formes et déterminer les coordonnées de leur centroïde,
   - Calculer la distance de ces centres par rapport au centre de l'anneau,
   - Déclencher le mécanisme de prise de photo en éclairage rasant si cette distance est inférieure au seuil fixé dans le programme (nous avons choisi 250 pixels).
     Ce mécanisme comporte les étapes suivantes :
     - Abaisser l'anneau,
     - Prendre une photo et la stocker dans un dossier (à choisir dans le programme),
     - Relever l'anneau.

   - Si la qualité de la vidéo acquise n'est pas satisfaisante, il existe également des boutons pour effectuer un traitement de l'image car le programme permettant de détecter les contours des formes fonctionne sur des images binarisées. (**Figure 14**)
     Pour améliorer la binarisation de la vidéo, il est possible d'effectuer
     - Une ouverture,
     - Une fermeture,
     - Un flou gaussien (pour lisser les contours),
     - Choisir une autre valeur de seuil pour la binarisation.

   - Pour optimiser la détection de contour il est également possible de définir
     - La taille maximale des formes à détecter,
     - La taille minimale des formes à détecter.

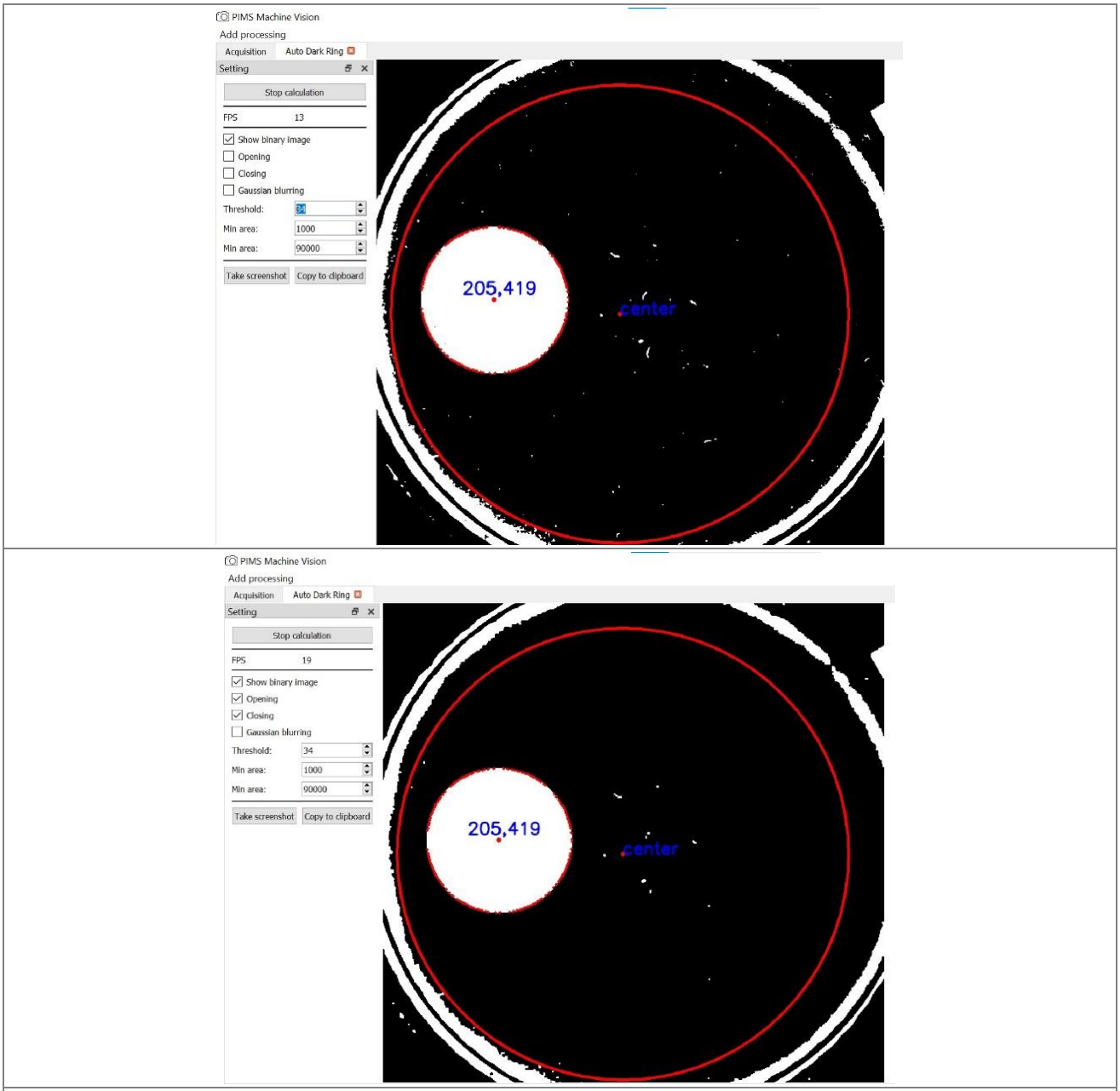


Figure 14 *: Captures d'écran de la fenêtre du bras articulé (**`Auto Dark Ring Tab`**) : visualisation de l'image binaire sans (en haut) et avec (en bas) opérations morphologiques*

4. Tests et Validation

4.1 Validation du contrôle manuel par écran tactile et par ordinateur

Après les différents réglages et ajustements effectués pour le dimensionnement mécanique du bras et après l'intégration de l'électronique sur la carte Disco, nous avons effectué la validation du contrôle manuel (par écran tactile et par ordinateur) du bras articulé avec l'anneau de LEDs sur le montage complet. (**Figure 15**)

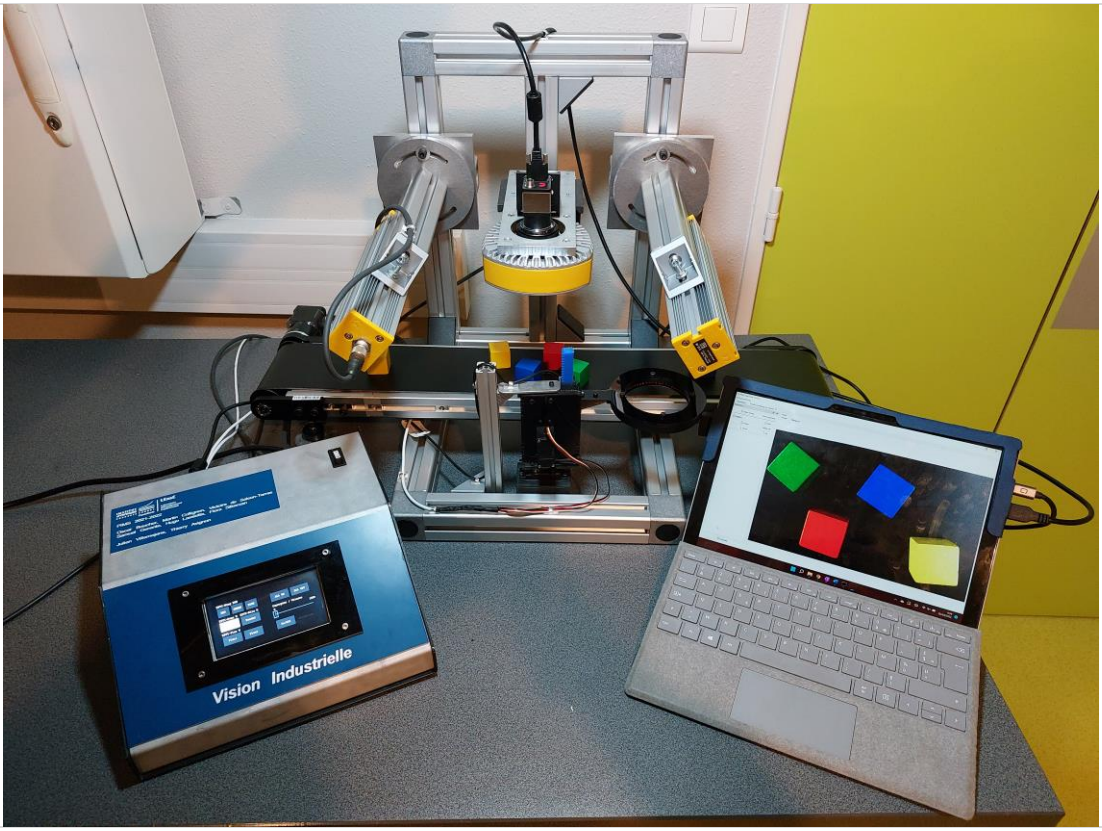


***Figure 15** : Photo d'ensemble du dispositif (PIMS & PROTIS)*

Pour cela, nous avons vérifié le bon fonctionnement de chaque bouton de commande dans les différentes fenêtres en se plaçant dans des configurations différentes.

Nous avons essentiellement rencontré des soucis d'ajustement de commandes à appliquer aux servomoteurs. En effet, pour certaines commandes, les servomoteurs dépassaient la limite de 90° et forçaient. Nous avons donc ajusté nos commandes pour éviter que ceux-ci ne forcent quelque-soit la situation.

Globalement, la validation du contrôle manuel, sur écran tactile ou sur ordinateur s'est déroulé sans difficulté majeure.

4.2 Validation du mode automatisé

Une fois le mode manuel validé, nous sommes passés à l'étape de validation du mode automatisé pour répondre à notre problématique de départ.

Nous avons rencontré deux problèmes principaux.

- Le premier problème a été un problème de communication avec la carte. En effet, les variables contenant les données de la communication série étaient définies dans le fichier main.py, et nous avons rencontré des difficultés à les importer dans le fichier AutoDarkRingTab.py correctement, en respectant la syntaxe des classes python. Finalement, en se plongeant dans la documentation en ligne, nous avons obtenu des réponses.

- Le deuxième problème a été d'avoir une bonne définition de la zone d'intérêt et de parvenir à réaliser un bon traitement de l'image (binarisation, ouverture, fermeture, flou gaussien).

  Dans un premier temps, plusieurs objets indésirables étaient détectés par le programme et provoquaient donc la mise en place non désirée de l'anneau.

  - En effet, comme convoyeur n'est pas uniformément noir, les nombreux petits défauts et reflets de l'éclairage sur celui-ci sont identifiés comme des petites formes, et lorsque celles-ci se rapprochent du centre de l'anneau, celui-ci se baisse et des photos sont prises.

    ➔ Pour résoudre ce problème, nous avons donc implémenté la possibilité pour l'utilisateur de régler la taille minimale des objets à détecter.

  - L'anneau de LED est également détecté comme une forme (dont le centre est par définition au centre de la zone d'intérêt) et provoque donc en permanence la mise en place du dispositif.

    ➔ Pour éviter ce dysfonctionnement, nous avons donc défini une zone d'intérêt circulaire (en rouge sur la **figure 16**) en dehors de laquelle aucune forme n'est prise en compte.

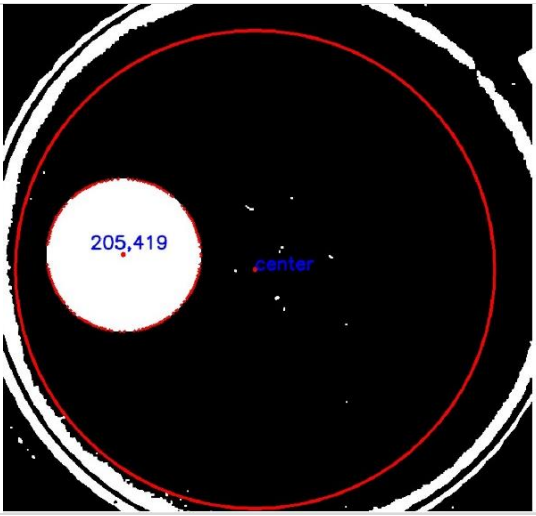


***Figure 16** : Visualisation de la zone d'intérêt circulaire (en rouge) en dehors de laquelle aucune forme n'est prise en compte*

Après toutes ces corrections et ces réglages, nous avons pu valider le mode automatisé de l'ensemble du dispositif.

5. Gestion du projet

5.1 Planning et organisation de l'équipe

Notre projet ProTIS a suivi le planning et l'organisation d'équipe suivants :

Etapes, actions, tâches,...	Qui	1 25-janv	2 08-févr	3 01-mars	4 08-mars	5 22-mars	6 28-mars	7 05-avr
Réponse à appel d'offre de la société SOLEC - Problématique	tous							
Cahier des charges	tous							
Schéma bloc 1	tous							
Constitution de l'équipe, écriture planning, répartition des tâches	tous							
1ère présentation du projet à la société SOLEC	tous							
Reprise en main de Solidworks	tous							
1er choix de conception pour le prototype : design du bras, positionnement, mouvement	tous							
Schéma fonctionnel	tous							
Choix du matériel : -2 moteurs à impulsion, source annulaire rasante monochrome (rouge)	tous							
MECANIQUE : Design des plans du bras sur Solidworks de la pièce du bas	Martin							
MECANIQUE : Design des plans du bras sur Solidworks de la pièce du dessus	Tong							
MECANIQUE : Design des plans du bras sur Solidworks de la pièce qui tient l'éclairage	Victoire, Albane							
Commande du matériel / se procurer le matériel	tous							
ELECTRONIQUE : Connexion avec la carte Nucléo qui gère les servomoteurs et pilotage des servomoteurs sur Mbed **- Tutoriel LenSE (contrôler un mouvement angulaire)** *Fonctionnement avec 2 cartes : une pour la commande de l'interface tactile qui contrôle éclairage et le convoyeur (PIMS- carte Disco) et une qui commande les servomoteurs(carte Nucléo)*	Victoire, Albane							
Conception : Translation verticale du bras (choix technologique) utilisation d'un servomoteur (adapté à notre besoin : petit déplacement)	tous							
Conception : Rotation du bras (choix technologique) - utilisation d'un moteur pas à pas (mise en place complexe --> **solution rejetée**) - utilisation d'un servomoteur : adapté à notre besoin (rotation de 90°) et mise en place plus simple	tous							
Réalisation du bras / assemblage provisoire avec l'anneau	tous							
Conception & tests / commande des servomoteurs : - Tests sur 2 servomoteurs indépendants du bras pour maîtriser leur angle de rotation (code Matlab)	tous							
Conception & test / commande translation et rotation du bras : - Test de la rotation et de la translation du bras en dehors du convoyeur (code Matlab) - Tests de la rotation et de la translation du bras sur le convoyeur (code Matlab) : adaptation du code aux contraintes du montage	tous							
Conception : choix technologique : Choix de 1 ou 2 cartes ? - solution retenue : **1 carte (carte Disco)** - commande du bras directe avec l'interface qui contrôle le convoyeur et les éclairages (nombre de ports de la carte Disco suffisant) **- Tutoriel LenSE (Comment échanger des données entre deux systèmes communicants ?)**	Victoire, Albane, Tong							
Modification de l'interface qui contrôle le convoyeur et les éclairages afin d'afficher 2 boutons qui permettent de contrôler le bras (le ranger ou le déployer)	Martin							

Etapes, actions, tâches,...	Qui	1	2	3	4	5	6	7
		25-janv	08-févr	01-mars	08-mars	22-mars	28-mars	05-avr
Electronique : Soudure des fils correspondants à l'alimentation des servomoteurs sur l'alimentation du boîtier. Soudure des fils de commande des servomoteurs aux broches D0 et D1 de la carte Disco.	Martin, Albane							
Essai (sur la carte Nucléo qui contrôle les servomoteurs) avec le nouveau bras imprimé, améliorations / avantages : - Rotation totale de 90 degrés - Pas de frottements	Victoire, Tong							
Conception : - Calcul de la correspondance entre l'angle de rotation du servomoteur et la translation verticale du bras - Calcul des correspondances entre les commandes et la distance de translation horizontale et l'angle de rotation du nouveau bras - Ajustement de ces commandes pour que les moteurs ne forcent pas	Victoire, Tong							
Essai avec le nouveau bras imprimé et la carte Disco définitive qui contrôle l'ensemble (convoyeur, interface, servomoteurs)	tous							
Conception : adaptation design bras --> Impression d'un nouveau bout du bras (les trous pour les vis sur l'extrémité du bras doivent correspondre aux trous sur l'anneau)	tous							
Conception : Création sur l'interface tactile de 3 boutons et d'un curseur afin de piloter manuellement le bras - Possibilité, grâce au boîtier de commande, de déployer ou replier le bras manuellement, d'allumer l'éclairage, de le baisser jusqu'en bas ou de le monter jusqu'en haut ou de régler sa position verticale de façon précise	Martin, Albane							
Conception / Tests & Réglages : - Détermination et affichage sur le boitier de la vitesse du tapis et du pourcentage correspondant pour le curseur (document vitesse convoyeur) pour pouvoir déterminer les temps à donner au code python pour l'automatisation du processus . - Détermination du temps disponible pour prendre la photo et baisser le bras tant que l'objet est dans le champ de la caméra délimité par l'intérieur de l'anneau	Martin, Albane, Tong							
Validation du fonctionnement par rapport au cahier des charges Validation du **contrôle manuel (écran tactile & ordinateur)**	Martin, Albane, Tong							
Automatisation du processus : - Une interface sur l'ordi pilotée par un programme en Python permet de faire la même chose qu'avec l'écran tactile (manuellement déployer ou replier le bras, allumer l'éclairage, le baisser jusqu'en bas ou le monter jusqu'en haut ou régler sa position verticale de façon précise) et d'effectuer un traitement des images	tous							
Automatisation du processus : - Communication avec la carte réussie. Amélioration du processus d'automatisation (choix de la distance entre le centre de la caméra et le centre de l'objet pour déclencher l'abaissement du bras, choix de la vitesse du tapis, ...) ---------- - Amélioration du traitement d'images obtenues par le processus automatisé (définir le seuil de niveaux de gris et la zone d'intérêt, régler le temps d'exposition et le nombre d'images par seconde...)	tous ---------- Victoire							
Validation du fonctionnement par rapport au cahier des charges Validation du **contrôle automatisé (ordinateur)** Identification des améliorations possibles	tous							
Préparation de la présentation commerciale	tous							
Installation et **Présentation commerciale**	tous							
Rédaction du rapport technique	tous							

5.2 Bilan de l'équipe sur ce projet

Ce projet a été une expérience très enrichissante non seulement d'un point gestion de projet et travail d'équipe, mais également d'un point de vue technique et sur les compétences acquises.

Lors de la première séance du projet, nous avons travaillé tous ensemble pour préciser le cahier des charges, le schéma bloc qui a ensuite donné lieu au schéma fonctionnel, le planning du projet et l'organisation de l'équipe. Cette étape a été fondamentale pour bien définir notre projet, nos objectifs communs et pour mettre en place une communication transparente et partagée à travers la plateforme collaborative Microsoft Teams. Après chaque séance, nous rédigions un petit compte-rendu sur un document partagé, ce qui permettait de faire un bilan de nos principales avancées mais également de pointer du doigt les problèmes techniques rencontrés en séance pour anticiper le travail à faire sur les prochaines étapes afin de tenir à la fois les objectifs que nous nous étions fixés dans le cahier des charges, ainsi que le planning.
Ensuite, nous avons fonctionné par binômes ou par trinôme ou seul sur certains problèmes, afin de paralléliser les tâches pour plus d'efficacité. Cependant, nous étions en général tous ensemble pour régler les problèmes plus complexes. Globalement, nous avons eu une excellente ambiance de travail au sein de notre équipe sans doute grâce à cette bonne communication que nous avons mise en place et grâce à l'écoute des uns des autres que nous avons eue, en étant attentifs aux remarques de chacun pour converger vers une solution optimale. Lorsqu'un problème délicat se présentait, nous avons su réfléchir ensemble et nous entraider. Nous ne nous sommes pas enfermés dans des rôles prédéfinis et une répartition des tâches trop rigide, nous avons préféré l'union, l'adaptabilité, la flexibilité et une grande transparence les uns avec les autres pour plus de créativité et d'efficacité.

D'un point de vue technique, ce projet nous a permis de tous travailler sur plusieurs métiers : mécanique, électronique et informatique. Cette pluridisciplinarité a été très formatrice. Nous avons également appris à piloter des servomoteurs et nous avons développé nos compétences en ce qui concerne l'échange de données entre deux systèmes communicants. Pour cela, nous avons utilisé deux langages de programmation : C et Python.
Par ailleurs, les tests et validations ont été progressifs avec des rebouclages réguliers sur la conception pour apporter des modifications, par petites touches, à la conception initiale et adapter les réglages nécessaires.

- ✓ Par exemple, sur la partie mécanique nous avons dû désigner un deuxième bras articulé pour régler les problèmes de frottements et de fixation du premier bras.
- ✓ Concernant la partie électronique, nous avons tout d'abord utilisé seulement une carte Nucléo pour contrôler les 2 servomoteurs (d'abord en dehors du convoyeur puis dans le montage global avec le convoyeur). Nous avons ensuite implémenté le tout dans la carte Disco (déjà utilisée sur PIMS) qui avait la capacité de réaliser l'ensemble des commandes (servomoteurs, éclairages et convoyeur) en présentant l'avantage de ne pas avoir à gérer en plus la communication entre les deux cartes.
- ✓ Pour la partie informatique, nous sommes partis des algorithmes de détection de forme de PIMS pour les adapter à l'éclairage rasant. Ceci a présenté l'avantage de ne pas repartir de zéro, mais également la contrainte de devoir intégrer de nouvelles fenêtres dans un programme initial plus vaste. Puis, il a fallu effectuer de nombreux réglages et calibrages pour arriver à un dispositif automatisé qui fonctionne dans différents contextes. En particulier, nous avons dû régler des

problèmes de communication entre l'ordinateur et la carte, puis avons dû adapter différents paramètres (comme le choix de la distance entre le centre de la caméra et le centre de l'objet pour déclencher l'abaissement du bras, le choix de la vitesse du tapis). Enfin il a fallu améliorer le traitement d'images en réglant d'autres paramètres (comme le seuil de niveaux de gris, la sélection de la zone d'intérêt, le réglage du temps d'exposition ou encore du nombre d'images par seconde de la caméra).

Ainsi, nous nous sommes coordonnés et nous avons avancé dans la même direction en unissant nos forces, nos aptitudes et nos compétences pour aboutir à une solution qui fonctionne et qui répond au cahier des charges fixé. Nous sommes très fiers de ce que nous avons accompli ensemble.

6. Conclusion

Ce projet s'est avéré instructif à la fois en termes de travail de groupe (sur le plan organisationnel et humain), mais aussi en termes d'innovation et de compétences acquises. Nous tenons à remercier les équipes du LEnsE, et en particulier M. Villemejane et M. Avignon pour leur aide et pour nous avoir permis de mener à bien ce projet au cours duquel nous avons tant appris.

Au terme de ce projet, nous avons réussi à obtenir ce que nous nous étions fixés au niveau du cahier des charges. Nous avons non seulement validé le contrôle manuel du bras articulé (optimisant l'éclairage rasant) par l'ordinateur et par l'écran tactile, mais également le mode automatisé par ordinateur qui permet en plus un traitement des images de la caméra en éclairage rasant.

Comme évolutions futures, afin d'automatiser plus finement et de systématiser le contrôle qualité (en vision industrielle) pour différentes tailles d'objets nécessitant un éclairage rasant, nous avons pensé à implémenter au niveau du convoyeur un capteur qui mesure précisément la taille de l'objet. Nous envisageons aussi d'implémenter d'autres algorithmes pour permettre d'autres traitements des images obtenues avec l'éclairage rasant, par exemple la détermination du pas de vis. Nous pourrions également ajouter un système de tri pour trier les pièces en sortie du convoyeur en fonction du choix de l'utilisateur.

Notre équipe reste évidemment à votre disposition pour toute question complémentaire.

7. Annexes

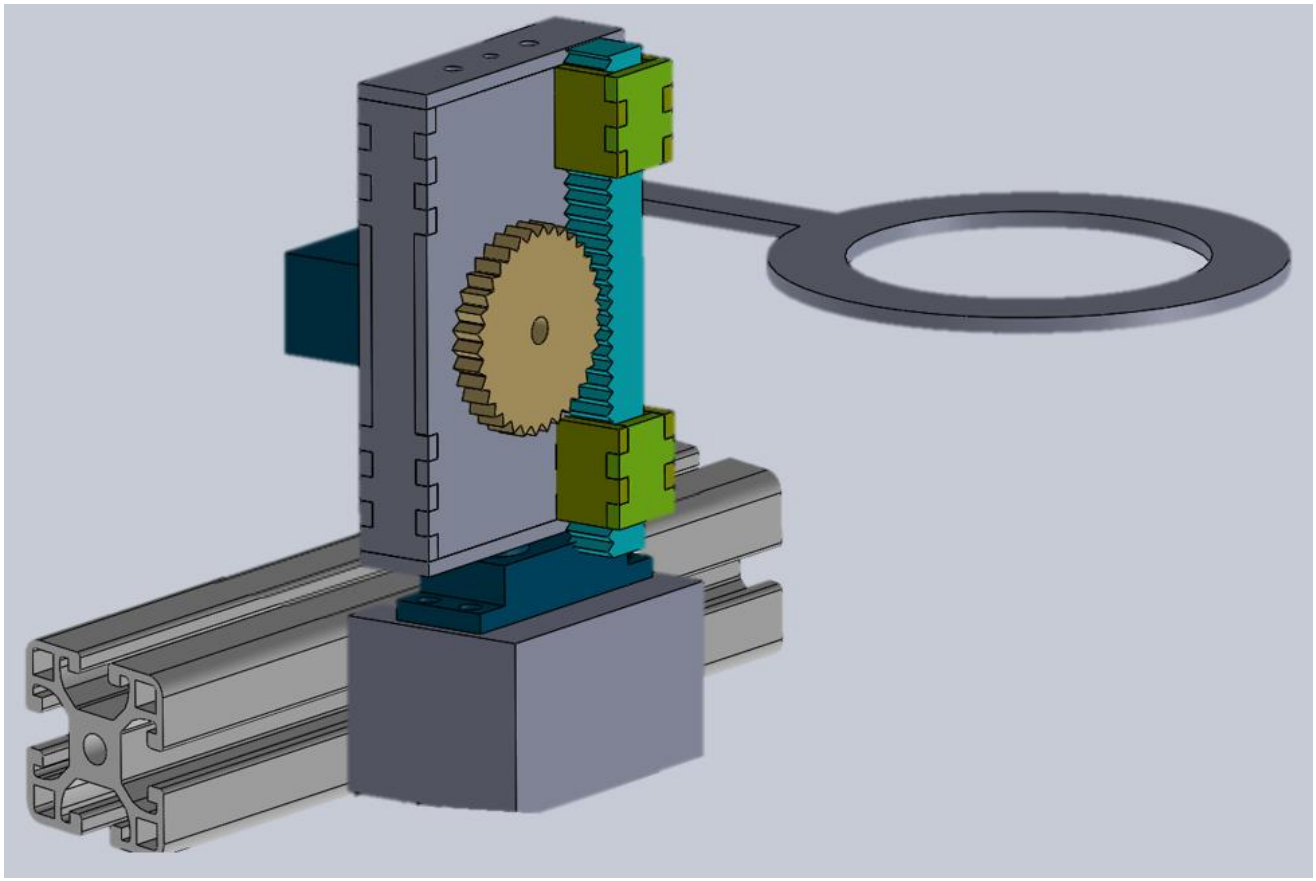
Annexe 1 : Bras articulé modélisé avec Solidworks

Annexe 2 : Code interface écran tactile

Annexe 3 : Code interface ordinateur

Annexe 1 :

Bras articulé modélisé avec Solidworks



Annexe 2 : Code interface écran tactile

main.ccp

```
/*****************************************************************************/
/*  Vision Industrielle IHM                                                  */
/*  main.ccp                                                                 */
/*****************************************************************************/
/*  LEnsE / Julien VILLEMEJANE       /   Institut d'Optique Graduate School */
/*****************************************************************************/
/*  PIMS Vision Industrielle :                                               */
/*      Oscar BOUCHER       Martin COLLIGNON       Victoire DE SALEON       */
/*      Samuel GERENTE      Hugo LASSIETTE         Flora SILBERZAN          */
/*****************************************************************************/
/*  ProTIS :                                                                 */
/*      Martin COLLIGNON    Albane LAPRAS                                    */
/*      Victoire de SALEON  Tong ZHU                                         */
/*****************************************************************************/
/*  PRINCIPE :                                                               */
/*  Le main initialise l'interface en mode ordinateur puis peut mettre à     */
/*  jour l'interface et faire une nouvelle initialisation si l'utilisateur   */
/*  veut changer de mode. Toutes ces actions dépendent du caractère reçu     */
/*  par la carte. Les actions sont séparées de 0,01s grâce à un ticker.      */
/*  Les caractères utilisés dans le main sont les suivants :                 */
/*      -   112 : mode Ordinateur                                            */
/*      -   113 : mode Ecran tactile                                         */
/*      -   114 : vérification que l'ordinateur est toujours connecté        */
/*              (Uniquement utilisé dans le mode Ordinateur)                 */
/*****************************************************************************/
/*  Library - Vision_Indus_IHM.cpp file                                      */
/*****************************************************************************/
/*  Testé sur DISCO-F746 / 22 Mars 2022                                      */
/*****************************************************************************/
/*  GUI library from :                                                       */
/*  2018/03/12, Copyright (c) 2018 MIKAMI, Naoki                             */
/*****************************************************************************/

#include "F746_GUI.hpp"
#include "Vision_Indus_IHM.h"

Serial      rs232(USBTX, USBRX);// Connexion between the card and the computer
char mode = 112;        // Initialization to computer mode
char pmode = 0;         // Previous mode
char data_received =0;  // Character recieved from the computer
Ticker ticker_data;     // Define the frequency of acquisition
Timer t;                // Timer to check if the computer is still connected
                        //(only in computer mode)

void chose_mode(void);
//_____________________________________________________________________//
//  Function that allows to :                                           //
//      - Change mode                                                   //
//      - Initialize the interface when the mode changes                //
//      - Update the interfaces following the mode the card is in       //
//  In the first part, it gets the data sent from the computer if it is //
//  readable.                                                           //
//  Then, it does the actions above following what data is received.    //
//  This function is attached to a ticker in the main that calls it     //
//  every 0.01s.                                                        //
//_____________________________________________________________________//

int main(){
    rs232.baud(115200); // Set the speed of the communication between the card
                        // and the computer
    initIHM(mode);      // Initialization to computer mode
    t.start();          // start the timer
    ticker_data.attach(&chose_mode, 0.01);  // Get data from the computer every
                                            // 0.01s and modify the mode
    while(1) {}
}
```

```
67
68
69  void chose_mode(void){
70      if(rs232.readable()){               // Stock the data if it is readable
71          data_received = rs232.getc();
72          rs232.putc(data_received);
73
74          if(data_received == 112){       // Computer mode
75              t.start();                  // Start the timer
76              mode = 112;
77          }
78
79  // 114 is sent regularly by the computer so that the card stays in computer mode
80          if(mode == 112 && data_received == 114)
81              t.reset();      // Reset timer because computer is still conencted
82
83  // LCD-screen mode
84          if(data_received == 113){
85              mode = 113;
86              t.stop();       // Stop and reset the timer
87              t.reset();      // (we don't need it in LCD-screen mode)
88          }
89
90  // If card in computer mode, do an action following the data received from it
91          if(mode == 112)
92              computer(data_received);
93
94      }
95
96  // Control if the computer is still sending data (>5s meeans we change mode)
97      if(mode == 112 && t.read()>5){
98          mode = 113;         // If not connected anymore go to LCD-screen mode
99          t.stop();           // Stop and reset the timer
100         t.reset();
101     }
102
103 // Initialize the interface if the mode is different from the previous one
104     if(pmode!=mode && (mode == 112 || mode == 113)) {
105         initIHM(mode);      // Initialization in the actual mode
106         pmode = mode;       // When mode changes, previous mode changes too
107     }
108
109 // Update the LCD-screen interface
110     if(mode == 113) {
111         updateIHM();        // Update when the user touches the screen
112     }
113 }
114
115
```

Vision_Indus_IHM.h

: Zones du code utilisées pour le projet ProTIS

```
/*****************************************************************************/
/*  Vision Industrielle IHM                                                  */
/*  Vision_Indus_IHM.h                                                       */
/*****************************************************************************/
/*  LEnsE / Julien VILLEMEJANE       /   Institut d'Optique Graduate School  */
/*****************************************************************************/
/*  PIMS Vision Industrielle :                                               */
/*      Oscar BOUCHER       Martin COLLIGNON       Victoire DE SALEON        */
/*      Samuel GERENTE      Hugo LASSIETTE         Flora SILBERZAN           */
/*****************************************************************************/
/*  ProTIS :                                                                 */
/*      Martin COLLIGNON    Albane LAPRAS                                    */
/*      Victoire de SALEON  Tong ZHU                                         */
/*****************************************************************************/
/*  Définition des variables globales, des entrées et sorties de la carte    */
/*  Disco et des fonctions utilisées dans Vision_Indus_IHM.ccp.              */
/*                                                                           */
/*  Les variables et différents objets définits dans Vision_Indus_IHM.h sont */
/*  expliquées plus en détails dans Vision_Indus_IHM.ccp.                    */
/*****************************************************************************/

#include "F746_GUI.hpp"
#include "mbed.h"

// Color ARGB (8-8-8-8)
#define     COLOR_OFF           0xFF666666  // Gris foncé
#define     CONV_DIR_FW         1           // Sens de rotation horaire (Forward)
#define     CONV_DIR_RV         0     // Sens de rotation anti-horaire (Reverse)


/* Entrées, sorties et variables pouvant être utilisées dans Vision_Indus_IHM.ccp */
// Flex
extern      DigitalOut      flex1_out;      // Barre de Leds 1
extern      DigitalOut      flex2_out;      // Barre de Leds 2
// Dome
extern      DigitalOut      dome_out;       // Dome
// Rasant
extern      DigitalOut      rasant_out;     // Eclairage annulaire rasant
// Effi-RGB
extern      DigitalOut      ringRGB_R_out;  // Leds rouges anneau
extern      DigitalOut      ringRGB_G_out;  // Leds vertes anneau
extern      DigitalOut      ringRGB_B_out;  // Leds bleues anneau


/* Eléments graphiques et variables */
extern      Label       title;
extern      Label       subtitle;
// All On-Off
extern      Button      light_on;
extern      Button      light_off;
extern      uint8_t     all_on_active;
// EFFI-Ring
extern      Label       roundRGB;
extern      Button      roundRGB_R;
extern      Button      roundRGB_G;
extern      Button      roundRGB_B;
extern      uint8_t     roundRGB_R_active;
extern      uint8_t     roundRGB_G_active;
extern      uint8_t     roundRGB_B_active;
// EFFI-Dome
extern      Label       dome;
extern      Button      dome_W;
extern      uint8_t     dome_active;
// EFFI-Rasant
extern      Label       rasant;         // Texte
extern      Button      rasant_R;       // Bouton
extern      uint8_t     rasant_active;  // 1 si allumé, 0 si éteint
// EFFI-Flex
extern      Label       flex1;
extern      Button      flex1_W;
extern      uint8_t     flex1_active;
extern      Button      flex2_W;
extern      uint8_t     flex2_active;
```

```
// Conveyor
extern      Label               conveyor;       // Texte
extern      NumericLabel<int>   conv_speed_lab; // Affichage vitesse
extern      SeekBar             conv_speed;     // Curseur
// Bras pour positionner l'éclairage rasant
extern      SeekBar             bras_pos;       // Curseur

/* Fonctions */
// Initialisation
void initIHM(char mode);
//_____________________________________________________________________//
//  Entrée :                                                           //
//      mode :  Chaîne de caractère associée à un mode de fonctionnement de la//
//          carte Disco (112 = mode "Ecran tactile"; 113 = mode "Ordinateur"  //
//  Principe :  Initialiser l'écran tactile de la carte et les variables selon//
//      le mode choisi par l'utilisateur. Fonction à utiliser à chaque fois   //
//      que l'utilisateur veut changer de mode.                            //
//_____________________________________________________________________//

// Mise à jour de l'interface de l'écran tactile
void updateIHM(void);
//_____________________________________________________________________//
//  Pas d'entrée ni de sortie.                                         //
//  Principe :  Mettre à jour l'interface lorsque la carte est en mode "Ecran //
//      tactile" et que l'utilisateur contrôle le dispositif avec l'écran     //
//      tactile. Une action est réalisée à chaque fois qu'un bouton ou un     //
//      curseur est touché.                                                //
//              Cette fonction contient une partie permettant l'ouverture d'un//
//      menu de contrôle du bras permettant de placer manuellement l'éclairage//
//      rasant. Ce menu s'ouvre lors de l'appui sur le bouton "Rasant".       //
//_____________________________________________________________________//

// Actions à realiser en fonction de la donnée reçue
void computer(char data_received);
//_____________________________________________________________________//
//  Entrée :                                                           //
//      data_received : chaîne de charactères correspondant à la donnée       //
//          envoyée par l'ordinateur et lue par la carte Disco                //
//  Principe :  Effectuer une action selon la donnée reçue (Allumer/Eteindre  //
//      un éclairage, changer la vitesse du convoyeur, son sens de rotation,  //
//      Deployer/Ranger le bras articulé et translater verticalement          //
//      l'éclairage rasant).                                               //
//              La liste des caractères utilisés ainsi que l'action qui leur  //
//      est associée est présente dans Vision_Indus_IHM.ccp.               //
//_____________________________________________________________________//

// Actions sur le convoyeur
void moveConveyor(char dir, int speed_us);
//_____________________________________________________________________//
//  Entrées :                                                          //
//      dir : chaîne de charactères correspondant au sens de rotation du      //
//          convoyeur (direction). CONV_DIR_FW pour le sens horaire (Forward);//
//          CONV_DIR_RV pour le sens anti-horaire (Reverse)                //
//      speed_us : entier correspondant à la période du signal PWM de commande//
//          du moteur pas à pas du convoyeur. Valeur en microseconde.         //
//  Principe :  Modifier la période du signal PWM pour changer la vitesse de  //
//      défilement du convoyeur et modifier le sens de défilement du convoyeur//
//      selon l'argument d'entrée.                                         //
//_____________________________________________________________________//
```

```
133
134  void stopConveyor(void);
135  //_____________________________________________________________________//
136  //  Pas d'entrée ni de sortie                                          //
137  //  Principe :  Arrêter le convoyeur en le désactivant et en rendant la//
138  //      commande nulle.                                                //
139  //_____________________________________________________________________//
140
141  int commande_rot(int angle);
142  //_____________________________________________________________________//
143  //  Entrée :                                                           //
144  //      angle : entier correspondant à l'angle dont on veut faire tourner le //
145  //          bras selon l'axe vertical. 0 bras rangé (parallèle au convoyeur  //
146  //          et à côté); 90 bras déplié (perpendiculeur au convoyeur, éclairage//
147  //          aligné avec la caméra).                                    //
148  //  Sortie : Entier correspondant à la durée du front montant du signal PWM  //
149  //      à appliquer pour atteindre l'angle souhaité. Valeur en microseconde. //
150  //  Principe :  Tourner selon l'axe verticale le bras qui supporte l'éclairage//
151  //  rasant de l'angle entré en argument.                               //
152  //_____________________________________________________________________//
153
154  int commande_HB(int angle);
155  //_____________________________________________________________________//
156  //  Entrée :                                                           //
157  //      angle : entier correspondant à l'angle dont on veut faire tourner le //
158  //          le servomoteur responsable de la translation. 0 correspond à la  //
159  //          position basse, 80 correspond à la position haute          //
160  //  Sortie : Entier correspondant à la durée du front montant du signal PWM  //
161  //      à appliquer pour atteindre l'angle souhaité. Valeur en microseconde. //
162  //  Principe :  Tourner le servomoteur pour le placer à l'angle souhaité   //
163  //      (compris entre 0° (bas) et 80° (haut)).                        //
164  //_____________________________________________________________________//
```

Vision_Indus_IHM.ccp

```
1  /*************************************************************************/
2  /*  Vision Industrielle IHM                                              */
3  /*  Vision_Indus_IHM.ccp                                                 */
4  /*************************************************************************/
5  /*  LEnsE / Julien VILLEMEJANE       /   Institut d'Optique Graduate School  */
6  /*************************************************************************/
7  /*  PIMS Vision Industrielle :                                           */
8  /*      Oscar BOUCHER       Martin COLLIGNON       Victoire DE SALEON    */
9  /*      Samuel GERENTE      Hugo LASSIETTE         Flora SILBERZAN       */
10 /*************************************************************************/
11 /*  ProTIS :                                                             */
12 /*      Martin COLLIGNON    Albane LAPRAS                                */
13 /*      Victoire de SALEON  Tong ZHU                                     */
14 /*************************************************************************/
15 /*  PRINCIPE :                                                           */
16 /*  Vision_Indus_IHM.ccp contient l'ensemble des fonctions qui permettent    */
17 /*  d'initialiser et de mettre à jour les interfaces dans les 2 modes        */
18 /*  (Ordinateur et Ecran tactile).                                       */
19 /*                                                                       */
20 /*  La première partie permet de mettre en place les éléments présents dans  */
21 /*  les interfaces et de définir l'ensemble des variables.               */
22 /*                                                                       */
23 /*  InitIHM : Permet d'initialiser l'interface selon le mode choisi. La      */
24 /*  fonction définit les éléments présents à l'écran ainsi que leur couleur. */
25 /*  Elle affecte aussi les bonnes valeurs initiales à toutes les variables   */
26 /*  utiles.                                                              */
27 /*                                                                       */
28 /*  updateIHM : Permet de mettre à jour l'interface dans laquelle la carte   */
29 /*  était à l'étape précédente. La fonction est composée de 2 parties :      */
30 /*      - la première met à jour l'interface de l'écran tactile lorsqu'un    */
31 /*      bouton est touché par l'utilisateur                              */
32 /*      - la seconde effectue une action sur les éclairages, le convoyeur    */
33 /*      ou le bras articulé selon le caractère envoyé par l'ordinateur est   */
34 /*      reçu par la carte (l'écran de la carte n'a pas besoin d'être mis à   */
35 /*      jour                                                             */
36 /*                                                                       */
37 /*  moveConveyor : permet de changer la vitesse et la diraction du convoyeur */
38 /*                                                                       */
39 /*  stopConveyor : permet d'arrêter le convoyeur                         */
40 /*                                                                       */
```

```
41  /*  commande_rot : permet de choisir l'angle de rotation du bras articulé     */
42  /*      selon l'axe verticale (angle compris entre 0° et 90°)                 */
43  /*                                                                            */
44  /*  commande_HB : permet de translater verticalement l'anneau selon l'angle   */
45  /*      correspondant (compris entre 0° et 80°)                               */
46  /*                                                                            */
47  /*  Les caractères (codés en ASCII) utilisés dans Vision_Indus_IHM.ccp sont   */
48  /*  les suivants :                                                            */
49  /*      -   de 0 à 50 : choisir la vitesse de rotation du tapis (à l'aide     */
50  /*              d'une loi reliant le nombre à la vitesse)                     */
51  /*      -   de 60 à 100 : choisir la position verticale de l'anneau (à        */
52  /*              l'aide d'une loi reliant le nombre à la vitesse)              */
53  /*      -   101 : allumer toutes les sources d'éclairage                      */
54  /*      -   102 : éteindre toutes les sources d'éclairage                     */
55  /*      -   103 : allumer/éteindre les leds rouges                            */
56  /*      -   104 : allumer/éteindre les leds vertes                            */
57  /*      -   105 : allumer/éteindre les leds bleues                            */
58  /*      -   106 : allumer/éteindre le dome                                    */
59  /*      -   107 : allumer/éteindre l'éclairage annulaire rasant               */
60  /*      -   108 : allumer/éteindre la barre de Leds 1                         */
61  /*      -   109 : allumer/éteindre ma barre de Leds 2                         */
62  /*      -   110 : changer la direction du convoyeur                           */
63  /*      -   111 : arrêter/démarrer la direction du convoyeur                  */
64  /*      -   115 : placer l'anneau en position haute                           */
65  /*      -   116 : placer l'anneau en position basse                           */
66  /*      -   117 : déployer (rotation de +90°)/ranger (position haute et       */
67  /*              rotation de -90°) le bras                                     */
68  /*                                                                            */
69  /*  Les caractères utilisés dans la partie ProTIS sont les suivants :         */
70  /*      De 0 à 50, de 60 à 100, 107, 110, 111, 115, 116, 117.
71  /******************************************************************************/
72
73  #include "Vision_Indus_IHM.h"
74
75
76  /*_______________________________Entrées et Sorties_______________________________*/
77  // Flex
78  DigitalOut      flex1_out(D4);
79  DigitalOut      flex2_out(D5);
80  // Dome
81  DigitalOut      dome_out(D11);
82  // Rasant
83  DigitalOut      rasant_out(D12);        // Commande pour l'éclairage rasant
84  // Effi-RGB
85  DigitalOut      ringRGB_R_out(D6);
86  DigitalOut      ringRGB_G_out(D7);
87  DigitalOut      ringRGB_B_out(D8);
88  // Conveyeur
89  PwmOut          conveyor_speed(D10);    // Commande pour la vitesse du convoyeur
90                                          // (selon la période du signal carré)
91  DigitalOut      conveyor_dir(D9);       // Commande pour la direction : 1 sens horaire, 0 sens anti-horaire
92  DigitalOut      conveyor_half(D13);     // Mode de fonctionnement du moteur
93  DigitalOut      conveyor_enable(D15);   // Activation du moteur
94  DigitalOut      conveyor_reset(D14);    // Réinitialisation des paramètres
95  DigitalIn       conveyor_fault(D2);     // Entrée pour signaler une erreur
96                                          // (pas utilisé)
97
98  PwmOut servo_mot_rot(D0);               // Commande pour le servomoteUR de la rotation du bras
99  PwmOut servo_mot_HB(D1);                // Commande pour le servomoteUR de la translation verticale du bras
100
101 /*_____________Création des éléments constituant les interfaces_____________*/
102 Label       mode_pc(240,120,"Mode Ordinateur",Label::CENTER, Font24);
103 // All On-Off
104 Button      light_on(265, 65, 80, 40, "ALL ON");
105 Button      light_off(355, 65, 80, 40, "ALL OFF");
106 uint8_t     all_on_active;
107 uint8_t     all_off_active;
```

```
// EFFI-Ring
Label       ringRGB(35, 50, "EFFI-Ring RGB", Label::LEFT);
Button      ringRGB_R(35, 65, 50, 40, "RED");
Button      ringRGB_G(95, 65, 50, 40, "GREEN");
Button      ringRGB_B(155, 65, 50, 40, "BLUE");
uint8_t     ringRGB_R_active;
uint8_t     ringRGB_G_active;
uint8_t     ringRGB_B_active;
// EFFI-Dome
Label       dome(35, 115, "EFFI-Dome W", Label::LEFT);
Button      dome_W(35, 130, 80, 40, "Dome");
uint8_t     dome_active;

/* EFFI-rasant */
Label       rasant(125, 115, "EFFI-Rlla R", Label::LEFT);   // Texte au-dessus du bouton
Button      rasant_R(125, 130, 80, 40, "Rasant");           // Bouton pour ouvrir le menu de contrôle du bras
uint8_t     rasant_active;      // Permet de savoir si l'éclairage rasanr est allumé ou non : 1 si ON, 0 si OFF

// EFFI-Flex
Label       flex1(35, 180, "EFFI-Flex W", Label::LEFT);
Button      flex1_W(35, 195, 80, 40, "Flex1");
uint8_t     flex1_active;
Button      flex2_W(125, 195, 80, 40, "Flex2");
uint8_t     flex2_active;

/* Convoyeur */
Label               conveyor(265, 120, "Convoyeur / Vitesse", Label::LEFT);     // Texte
NumericLabel<int>   conv_speed_lab(405, 120, "V = ", Label::LEFT);              // Affichage de la vitesse du convoyeur
SeekBar             conv_speed(270, 165, 170, 0, 100, 0, "0", "", "100%");      // Curseur pour régler la vitesse
NumericLabel<int>   x_slide(345,136,"V = ",Label::CENTER);                      // Affichage du pourcentage
Button      conv_direction(355, 195, 80, 40, "Direction");                      // Bouton "Direction"
Button      conv_stop(265, 195, 80, 40, "On/Off");  // Bouton "On/Offf" pour mettre en marche ou arrêter le convoyeur
uint8_t     conv_stop_active;       // Permet de savoir si le convoyeur est en marche ou non : 1 si actif, 0 si à l'arrêt

uint8_t     conv_direction_active;  // Permet de savoir dans quel sens est le convoyeur : 1 sens horaire, 0 sens anti-horaire

int         vitesse;    // Vitesse du convoyeur
int         periode;    // Période du signal PWM de commande

/* Interface du menu pour contrôler bras, éclairage rasant et convoyeur */
Label       rasant_2(125, 50, "Rasant", Label::LEFT);               // Texte "Rasant" au-dessus du bouton suivant
Button      rasant_on(125, 65, 80, 40, "ON / OFF");                 // Bouton pour allumer ou éteindre l'eclairage rasant
Label       deplie(35, 50,"Deplier/Ranger",Label::LEFT);            // Texte "Deplier/Ranger" au-dessus du bouton suivant
Button      rasant_turn_B(35, 65, 80, 40);                          // Bouton pour déplier ou ranger le bras
Button      haut(35, 130, 80, 40, "Haut");                          // Bouton pour placer le bras au plus haut
Button      bas(125, 130, 80, 40, "Bas");                           // Bouton pour placer le bras au plus bas (posé sur le convoyeur)
Label       position(35, 180, "Position du bras", Label::LEFT);     // Texte au-dessus du curseur
NumericLabel<int>   position_value(175, 180, "z = ", Label::LEFT);  // Affichage de la position verticale
SeekBar             bras_pos(35, 225, 170, 0, 100, 0, "0", "", "100%");  // Curseur pour changer la position verticale de l'éclairage
NumericLabel<int>   z_slide(110,196,"z = ",Label::CENTER);          // Affichage du pourcentage du curseur
Button              close(425, 30, 20,20,"X");                      // Bouton pour fermer le menu de contrôle du bras

int         angle;          // Angle du servomoteur responsable de la transition verticale
int         hauteur;        // Hauteur de l'éclairage rasant par rapport au convoyeur
int         deplier;        // Permet de savoir si le bras est déplié ou non : 1 si déplié, 0 si rangé
uint8_t     turn_active;    // 1 si rangé, 0 si déplié

//_____________Initialisation de l'interface selon le mode choisi_____________//

void initIHM(char mode){
// EFFI-Ring
    ringRGB_R_active = 0;
    ringRGB_R_out = 1;
    ringRGB_G_active = 0;
    ringRGB_G_out = 1;
    ringRGB_B_active = 0;
    ringRGB_B_out = 1;
// EFFI-Dome
    dome_active = 0;
    dome_out = 0;
// EFFI-Rlla
    rasant_active = 0;  // Initialisation : Eclairage rasant éteint
    rasant_out = 0;
```

```
// EFFI-Flex
    flex1_active = 0;
    flex1_out = 1;
    flex2_active = 0;
    flex2_out = 1;
// Convoyeur
    int8_t x = (int8_t)conv_speed.GetValue();       // Obtenir le pourventage du curseur
    vitesse = (0.0054/((2000-7*x)*1e-6)-0.0033)*10; // Calcul de la vitesse avec une loi expérimentale
    conv_speed_lab.Draw("%3d mm/s", vitesse);       // Affichage de la vitesse
    x_slide.Draw("%3d %%",x);                       // Affichage du pourcentage
    // Initialisation du moteur du convoyeur
    conveyor_enable = 0;                            // Commandes à envoyer pour faire fonctionner le moteur
    conveyor_reset = 1;
    conveyor_half = 1;
    conveyor_speed.period_us(1000);                 // Période du signal PWM initialisée à 1 ms
    conveyor_dir = CONV_DIR_FW;                     // Sens horaire (forward)
    conv_direction_active = 1;                      // Convoyeur dans le sens horaire
    conveyor_speed.write(0);                        // Rapport cyclique nul donc commande nulle
    conv_stop_active = 0;                           // Convoyeur à l'arrêt


// servo moteur contrôlant la rotation du bras
    servo_mot_rot.period_ms(20);                    // Initialisation période

// servo moteur ajustant la hauteur de l'éclairage rasant
    servo_mot_HB.period_ms(20);                     // Initialisation période

    servo_mot_HB.pulsewidth_us(commande_HB(80));    // Position haute
    wait(0.3f);
    servo_mot_rot.pulsewidth_us(commande_rot(0));   // Rangement du bras

// Initialisation du mode "Ecran tactile"
    if(mode == 113) {
        BSP_LCD_Clear(0xFF003538);
        ringRGB.Draw(0xFFFFFFFF);
        dome.Draw(0xFFFFFFFF);
        rasant.Draw(0xFFFFFFFF);
        flex1.Draw(0xFFFFFFFF);
        conveyor.Draw(0xFFFFFFFF);
        SeekBar       conv_speed(255+a, 165, 170, 0, 100, 0, "0", "", "100%");
        all_on_active = 0;
        light_off.Draw(0xFFFFFFFF, 0xFF000000);
        light_on.Draw(COLOR_OFF, 0xFFFFFFFF);
        ringRGB_R.Draw(COLOR_OFF);
        ringRGB_G.Draw(COLOR_OFF);
        ringRGB_B.Draw(COLOR_OFF);
        dome_W.Draw(COLOR_OFF);
        rasant_R.Draw(COLOR_OFF);
        flex1_W.Draw(COLOR_OFF);
        flex2_W.Draw(COLOR_OFF);
        conv_direction.Draw(0xFF333333,0xFF555555);
        conv_stop.Draw(COLOR_OFF);
    }

// Initialisation du mode "Ordinateur"
    if(mode == 112){                // Réinitialisation de l'écran et affichage du texte "Mode Ordinateur"
        BSP_LCD_Clear(0xFF003538);
        mode_pc.Draw(0xFFFFFFFF);
        turn_active = 1;
        periode = 2000;
    }
}

//_____Mise à jour de l'interface de l'écran et actions sur les éclairages____//
void updateIHM(void){
    if (light_on.Touched()) {    // Press the 'ALL ON' button
        all_on_active = 1;
        if(all_on_active == 1) { // Switch on all the sources
            light_off.Draw(COLOR_OFF, 0xFFFFFFFF);
            light_on.Draw(0xFFFFFFFF, 0xFF000000);

            ringRGB_R_active = 1;
            ringRGB_R_out = 0;
            ringRGB_R.Draw(0xFFFF0000);
            ringRGB_G_active = 1;
            ringRGB_G_out = 0;
            ringRGB_G.Draw(0xFF33DD33, 0xFF000000);
            ringRGB_B_active = 1;
            ringRGB_B_out = 0;
            ringRGB_B.Draw(0xFF0000FF);

            dome_active = 1;
            dome_out = 1;
            dome_W.Draw(0xFFFFFFFF, 0xFF000000);

            rasant_active = 1;
            rasant_out = 1;
            rasant_R.Draw(0xFFFF0000);

            flex1_active = 1;
            flex1_out = 0;
```

```
            flex1_W.Draw(0xFFEEEEEE, 0xFF000000);
            flex2_active = 1;
            flex2_out = 0;
            flex2_W.Draw(0xFFEEEEEE, 0xFF000000);
            all_on_active = 0;

            wait(0.2f);
        }
    }
    else {
        if (light_off.Touched()) {       // Press the 'ALL OFF' button
            all_off_active = 1;
            if(all_off_active == 1) {    // Switch off all the sources
                light_on.Draw(COLOR_OFF, 0xFFFFFFFF);
                light_off.Draw(0xFFFFFFFF, 0xFF000000);

                ringRGB_R_active = 0;
                ringRGB_R_out = 1;
                ringRGB_R.Draw(COLOR_OFF);
                ringRGB_G_active = 0;
                ringRGB_G_out = 1;
                ringRGB_G.Draw(COLOR_OFF, 0xFFFFFFFF);
                ringRGB_B_active = 0;
                ringRGB_B_out = 1;
                ringRGB_B.Draw(COLOR_OFF);

                dome_active = 0;
                dome_out = 0;
                dome_W.Draw(COLOR_OFF, 0xFFFFFFFF);

                rasant_active = 0;
                rasant_out = 0;
                rasant_R.Draw(COLOR_OFF);

                flex1_active = 0;
                flex1_out = 1;
                flex1_W.Draw(COLOR_OFF, 0xFFFFFFFF);
                flex2_active = 0;
                flex2_out = 1;
                flex2_W.Draw(COLOR_OFF, 0xFFFFFFFF);

                all_off_active = 0;
            }
            wait(0.2f);
        }

        if (ringRGB_R.Touched()) {   // Press the 'RED' button
            if(ringRGB_R_active) {
                light_on.Draw(COLOR_OFF, 0xFFFFFFFF);
                ringRGB_R_active = 0;
                ringRGB_R_out = 1;
                all_on_active = 0;
                ringRGB_R.Draw(COLOR_OFF);
            } else {
                light_off.Draw(COLOR_OFF, 0xFFFFFFFF);
                ringRGB_R_active = 1;
                ringRGB_R_out = 0;
                ringRGB_R.Draw(0xFFFF0000);
            }
            wait(0.2f);

        }
        if (ringRGB_G.Touched()) {    // Press the 'GREEN' button
            if(ringRGB_G_active) {
                light_on.Draw(COLOR_OFF, 0xFFFFFFFF);
                ringRGB_G_active = 0;
                ringRGB_G_out = 1;
                all_on_active = 0;
                ringRGB_G.Draw(COLOR_OFF, 0xFFFFFFFF);
            } else {
                light_off.Draw(COLOR_OFF, 0xFFFFFFFF);
                ringRGB_G_active = 1;
                ringRGB_G_out = 0;
                ringRGB_G.Draw(0xFF33DD33, 0xFF000000);
            }
            wait(0.2f);
        }
        if (ringRGB_B.Touched()) {      // Press the 'BLUE' button
            if(ringRGB_B_active) {
                light_on.Draw(COLOR_OFF, 0xFFFFFFFF);
                ringRGB_B_active = 0;
                ringRGB_B_out = 1;
                all_on_active = 0;
                ringRGB_B.Draw(COLOR_OFF);
            } else {
                light_off.Draw(COLOR_OFF, 0xFFFFFFFF);
                ringRGB_B_active = 1;
                ringRGB_B_out = 0;
                ringRGB_B.Draw(0xFF0000FF);
            }
            wait(0.2f);
```

```
        }
        if (dome_W.Touched()) {         // Press the 'Dome' button
            if(dome_active) {
                light_on.Draw(COLOR_OFF, 0xFFFFFFFF);
                dome_active = 0;
                dome_out = 0;
                all_on_active = 0;
                dome_W.Draw(COLOR_OFF, 0xFFFFFFFF);
            } else {
                light_off.Draw(COLOR_OFF, 0xFFFFFFFF);
                dome_active = 1;
                dome_out = 1;
                dome_W.Draw(0xFFFFFFFF, 0xFF000000);
            }
            wait(0.2f);
        }

/*_____________Début de la partie menu "Eclairage rasant"_____________*/
        if (rasant_R.Touched()) {       // Si appui sur le bouton "Rasant"
// Initialisation de l'interface permettant le contrôle du bras
            rasant_R.Draw(0xFFFF0000);                  // Bouton en rouge
            wait(0.1f);                                 // Attendre 0,1s
            BSP_LCD_Clear(0xFF003538);                  // Réinitialisation de l'écran
    //_______Affichage des éléments du menu_______//
            BSP_LCD_DrawRect(15,15,450,240);            // Grand rectangle
            conveyor.Draw(0xFFFFFFFF);                  // Affichage du texte et du curseur vitesse
            SeekBar       conv_speed(270, 165, 170, 0, 100, 0, "0", "", "100%");
            all_on_active = 0;                          // Les éclairages ne sont pas tous allumés
            conv_direction.Draw(0xFF333333,0xFF555555); // Fond gris foncé, texte gris clair
            conv_stop.Draw(COLOR_OFF);                  // Gris foncé
            SeekBar     bras_pos(35, 225, 170, 0, 100, 0, "0", "", "100%"); // Affichage du curseur position
            rasant_2.Draw(0xFFFFFFFF);                  // texte blanc
            close.Draw(0xFFFF0000);                     // Fond rouge
            rasant_on.Draw(0xFF333333,0xFF777777);      // Fond gris foncé, texte gris clair
            rasant_turn_B.Draw(COLOR_OFF);              // Gris foncé
            position.Draw(0xFFFFFFFF);                  // Texte blanc
            haut.Draw(0xFF333333,0xFF777777);           // Fond gris foncé, texte gris clair
            bas.Draw(0xFF333333,0xFF777777);            // Fond gris foncé, texte gris clair
            rasant_active = 1;
            turn_active = 1;                            // Bras rangé
            deplie.Draw(0xFFFFFFFF);
            servo_mot_HB.pulsewidth_us(commande_HB(80));    // Eclairage placé en haut
            wait(0.2f);
            servo_mot_rot.pulsewidth_us(commande_rot(0));   // Bras rangé
            while(1){
                if(rasant_turn_B.Touched()) {           // Si appui sur bouton "Deplier/Ranger"
                    if(turn_active) {                       // Et si le bras est rangé alors dépliement du bras
                        turn_active = 0;                        // Changement de valeur pour ranger le bras au prochain appui
                        deplier = 1;                            // Le bras est déplié, permet d'accéder aux autres actions
                        rasant_turn_B.Draw(0xFFFFFFFF);         // Changements des couleurs pour montrer que les boutons
                        wait(0.2f);                             // ne sont pas utilisables
                        rasant_turn_B.Draw(0xFF0e89b4);
                        haut.Draw(COLOR_OFF, 0xFFFFFFFF);
                        bas.Draw(COLOR_OFF, 0xFFFFFFFF);
                        rasant_on.Draw(COLOR_OFF, 0xFFFFFFFF);
                        servo_mot_rot.pulsewidth_us(commande_rot(90)); // dépliement du bras
                    } else {                                // Sinon rangement du bras
                        turn_active = 1;                        // Changement de valeur pour deplier le bras au prochain appui
                        deplier = 0;                            // Le bras est rangé, empêche l'accès aux autres actions
                        rasant_turn_B.Draw(0xFFFFFFFF);         // Changements des couleur pour montrer que les boutons
                        wait(0.2f);                             // sont utilisables
                        rasant_turn_B.Draw(COLOR_OFF);
                        haut.Draw(0xFF333333,0xFF777777);
                        bas.Draw(0xFF333333,0xFF777777);
                        rasant_on.Draw(0xFF333333,0xFF777777);
                        rasant_active = 0;                      // Eclairage éteint
                        rasant_out = 0;
                        servo_mot_HB.pulsewidth_us(commande_HB(80));   // Retour en position haute
                        wait(0.3f);
                        servo_mot_rot.pulsewidth_us(commande_rot(0));  // Rotation pour ranger le bras
                    }
                    wait(0.2f);
                }
                if(deplier == 1 && rasant_on.Touched()) {   // Si bras déplié et appui sur bouton "ON / OFF"
                    if(rasant_active) {                     // Et si l'éclairage rasant est allumé
                        rasant_active = 0;                      // Changement de valeur pour allumer au prochain appui
                        rasant_out = 0;                         // Eteindre
                        rasant_on.Draw(COLOR_OFF);
                        wait(0.2f);                             // Attendre 0,2s
                    } else {                                // Sinon
                        rasant_active = 1;                      // Changement de valeur pour éteindre au prochain appui
                        rasant_out = 1;                         // Allumer
                        rasant_on.Draw(0xFFFF0000);             // Bouton en rouge
                        wait(0.2f);                             // Attendre 0,2s
                    }
                }

                if(deplier == 1 && haut.Touched()){         // Si bras déplié et appui sur bouton "Haut"
                    haut.Draw(0xFFFFFFFF, 0xFF000000);          // Fond blanc, texte noir
                    bas.Draw(COLOR_OFF, 0xFFFFFFFF);            // Fond gris foncé, texte blanc
                    servo_mot_HB.pulsewidth_us(commande_HB(80));// 80° pour placer l'éclairage en position haute
```

```
454                     wait(0.2f);
455                 }
456                 if(deplier == 1 && bas.Touched()){          // Si bras déplié et appui sur bouton "Bas"
457                     bas.Draw(0xFFFFFFFF, 0xFF000000);           // Fond blanc, texte noir
458                     haut.Draw(COLOR_OFF, 0xFFFFFFFF);           // Fond gris foncé, texte blanc
459                     servo_mot_HB.pulsewidth_us(commande_HB(0)); // 0° pour placer l'éclairage en position basse
460                     wait(0.2f);
461                 }
462                 if (deplier == 1 && bras_pos.Slide()) {     // Si bras déplié et déplacement du curseur position
463                     haut.Draw(COLOR_OFF, 0xFFFFFFFF);           // Fond gris foncé, texte blanc
464                     bas.Draw(COLOR_OFF, 0xFFFFFFFF);            // Fond gris foncé, texte blanc
465                     int8_t z = (int8_t)bras_pos.GetValue();     // Pourcentage du curseur
466                     angle = (int) 80*z/100;                     // Calcul de l'angle que doit avoir le sercomoteur
467                     hauteur = (int) angle/2.85;                 // Calcul de la hauteur de l'éclairage
468                     position_value.Draw("%3d mm", hauteur);     // Affichage de la hauteur
469                     z_slide.Draw("%3d %%",z);                   // Affichage du pourcentage
470                     servo_mot_HB.pulsewidth_us(commande_HB(angle)); // Translation vertivale de l'éclairage selon l'angle
471                 }
472                 // Possibilité de contrôler le convoyeur
473                 if (conv_speed.Slide()) {                   // Si déplacement du curseur vitesse convoyeur
474                     int8_t x = (int8_t)conv_speed.GetValue();   // Pourcentage du curseur
475                     vitesse = 0.0024*x*x+0.0107*x+27.355;       // Calcul de la vitesse
476                     conv_speed_lab.Draw("%3d mm/s", vitesse);   // Affichage de la vitesse
477                     x_slide.Draw("%3d %%",x);                   // Affichage du pourcentage
478                     if(conveyor_enable == 1) {                  // Si le convoyeur est en marche
479                         if(conv_direction_active == 0)              // Et si sens anti-horaire
480                             moveConveyor(CONV_DIR_RV,floor(2093.8-10.475*x));   // Changement de vitesse
481                         else                                        // Sinon
482                             moveConveyor(CONV_DIR_FW,floor(2093.8-10.475*x));   // Changement de vitesse
483                     }
484                 }
485                 if(conv_stop_active == 1) {                 // Si le convoyeur est en marche
486                     if (conv_direction.Touched()) {             // et si appui sur le bouton "Direction"
487                         conv_direction.Draw(0xFFffca4f,0xFF000000); // Fond orange, texte noir
488                         stopConveyor();                             // Arrêt du convoyeur pendant 0,2s
489                         wait(0.2f);
490                         if(conv_direction_active==1) {              // Si sens horaire
491                             conv_direction_active = 0;                  // Changement de valeur pour changer de sens au prochain appui
492                             int8_t x = (int8_t)conv_speed.GetValue();   // Pourcentage du curseur
493                             moveConveyor(CONV_DIR_RV,floor(2093.8-10.475*x));  // Changement de sens, même vitesse
494                         } else {                                    // Sinon
495                             conv_direction_active = 1;                  // Changement de valeur pour changer de sens au prochain appui
496                             int8_t x = (int8_t)conv_speed.GetValue();   // Pourcentage du curseur
497                             moveConveyor(CONV_DIR_FW,floor(2093.8-10.475*x));  // Changement de sens, même vitesse
498                         }
499                     } else                                      // S'il est arrêté
500                         conv_direction.Draw(0xFFFFB200,0xFF000000); // Fond orange, texte noir
501                 }
502                 if (conv_stop.Touched()) {                  // Si appui sur le bouton "on/off"
503                     if(conv_stop_active==1) {                   // Et si le convoyeur est en marche
504                         conv_stop_active = 0;                       // Changement de valeur pour mettre en marche au prochain appui
505                         stopConveyor();                             // Arrêt du convoyeur
506                         conv_stop.Draw(COLOR_OFF, 0xFFFFFFFF);      // Fond gris foncé, texte blanc
507                         conv_direction.Draw(0xFF333333,0xFF777777); // Fond gris foncé, texte gris
508                     } else {                                    // Sinon
509                         conv_stop_active = 1;                       // Changement de valeur pour arrêter au prochain appui
510                         if(conv_direction_active==1) {              // Si sens horaire
511                             int8_t x = (int8_t)conv_speed.GetValue();
512                             moveConveyor(CONV_DIR_FW,floor(2093.8-10.475*x)); // Même sens et vitesse correspondant au % du curseur
513                         } else {
514                             int8_t x = (int8_t)conv_speed.GetValue();
515                             moveConveyor(CONV_DIR_RV,floor(2093.8-10.475*x)); // Même sens et vitesse correspondant au % du curseur
516                         }
517                         conv_stop.Draw(0xFFFFFFFF, 0xFF000000);     // Fond blanc, texte noir
518                         conv_direction.Draw(0xFFFFB200,0xFF000000); // Fond orange, texte noir
519                     }
520                     wait(0.2f);
521                 }
522                 if(close.Touched()){                        // Si appui sur le bouton "X"
523                     deplier = 0;                                // Bras rangé
524                     rasant_active = 0;                          // Eclairage éteint
525                     rasant_out = 0;
526                     servo_mot_HB.pulsewidth_us(commande_HB(80));// Eclairage en position haute
527                     wait(0.3f);
528                     servo_mot_rot.pulsewidth_us(commande_rot(0));// Repliement du bras
529                     initIHM(113);                                // Retour au menu général de contrôle des eclairages
530                     break;                                       // et du convoyeur
531                     }
532                 }
533         }
534 /*__________________Fin de la partie menu "Eclairage rasant"_________________*/
535
536         if (flex1_W.Touched()) {        // Press the 'Flex1' button
537             if(flex1_active) {
538                 light_on.Draw(COLOR_OFF, 0xFFFFFFFF);
539                 flex1_active = 0;
540                 flex1_out = 1;
541                 all_on_active = 0;
542                 flex1_W.Draw(COLOR_OFF, 0xFFFFFFFF);
543             } else {
544                 light_off.Draw(COLOR_OFF, 0xFFFFFFFF);
```

```
                flex1_active = 1;
                flex1_out = 0;
                flex1_W.Draw(0xFFEEEEEE, 0xFF000000);
            }
            wait(0.2f);
        }

        if (flex2_W.Touched()) {        // Press the 'Flex2' button
            if(flex2_active) {
                light_on.Draw(COLOR_OFF, 0xFFFFFFFF);
                flex2_active = 0;
                flex2_out = 1;
                all_on_active = 0;
                flex2_W.Draw(COLOR_OFF, 0xFFFFFFFF);
            } else {
                light_off.Draw(COLOR_OFF, 0xFFFFFFFF);
                flex2_active = 1;
                flex2_out = 0;
                flex2_W.Draw(0xFFEEEEEE, 0xFF000000);
            }
            wait(0.2f);
        }
    }
    if (conv_speed.Slide()) {        // Move the cursor of the seekbar
        int8_t x = (int8_t)conv_speed.GetValue();
        vitesse = 0.0024*x*x+0.0107*x+27.355; // Computation of the speed from the position of the cursor
        conv_speed_lab.Draw("%3d mm/s", vitesse);
        x_slide.Draw("%3d %%",x);
        if(conveyor_enable == 1) {
            if(conv_direction_active == 0)
                moveConveyor(CONV_DIR_RV,floor(2093.8-10.475*x));
            else
                moveConveyor(CONV_DIR_FW,floor(2093.8-10.475*x));
        }
    }

    if(conv_stop_active == 1) {         // Enables to click the 'Direction' button only if the conveyor is on
        if (conv_direction.Touched()) {      // Press the 'Direction' button
            conv_direction.Draw(0xFFffca4f,0xFF000000);
            stopConveyor();
            wait(0.2f);
            if(conv_direction_active==1) { // Depends on the previous direction
                conv_direction_active = 0;
                int8_t x = (int8_t)conv_speed.GetValue();
                moveConveyor(CONV_DIR_RV,floor(2093.8-10.475*x));
            } else {
                conv_direction_active = 1;
                int8_t x = (int8_t)conv_speed.GetValue();
                moveConveyor(CONV_DIR_FW,floor(2093.8-10.475*x));
            }
        } else
            conv_direction.Draw(0xFFFFB200,0xFF000000);
    }

    if (conv_stop.Touched()) {         // Press the 'On/Off' button
        if(conv_stop_active==1) {
            conv_stop_active = 0;
            stopConveyor();
            conv_stop.Draw(COLOR_OFF, 0xFFFFFFFF);
            conv_direction.Draw(0xFF333333,0xFF777777);
        } else {
            conv_stop_active = 1;
            if(conv_direction_active==1) {  // Depends on the previous direction
                int8_t x = (int8_t)conv_speed.GetValue();
                moveConveyor(CONV_DIR_FW,floor(2093.8-10.475*x));
            } else {
                int8_t x = (int8_t)conv_speed.GetValue();
                moveConveyor(CONV_DIR_RV,floor(2093.8-10.475*x));
            }
            conv_stop.Draw(0xFFFFFFFF, 0xFF000000);
            conv_direction.Draw(0xFFFFB200,0xFF000000);
        }
        wait(0.2f);
    }
}

//__________Effectuer une action selon la donnée reçue par la carte__________//

void computer(char data_received)
{
// Données permettant de changer la vitesse du convoyeur
    if(data_received<=50) {             // Si la donnée reçue est entre 0 et 50
        periode = floor(2093.8-data_received*10.475); // La convertir en la période à donner au signal PWM pour changer la vitesse
        if(conveyor_enable == 1) {      // Si convoyeur actif
            if(conv_direction_active == 0)  // Et selon le sens de rotation du convoyeur
                moveConveyor(CONV_DIR_RV,periode);  // Affecter la période au signal PWM
            else
                moveConveyor(CONV_DIR_FW,periode);  // Idem dans l'autre sens
        }
    }
// Données permettant de changer la hauteur de l'éclairage rasant
```

```
    if(data_received>=60 && data_received<=100){        // Si la donnée reçue est entre 60 et 100
        angle = 2*(data_received-60);                   // Calcul de l'angle à fournir au servomoteur responsable de la translation
        servo_mot_HB.pulsewidth_us(commande_HB(angle));     // Placer l'éclairage à la hauteur correspondant à l'angle calculé
    }
// Utilisation d'un switch pour lister les actions à réaliser selon le caractère reçu (codé en ASCII)
    switch(data_received) {
        case 101:                       // All the lights on
            all_on_active = 1;
            if(all_on_active == 1) {
                ringRGB_R_active = 1;
                ringRGB_R_out = 0;
                ringRGB_G_active = 1;
                ringRGB_G_out = 0;
                ringRGB_B_active = 1;
                ringRGB_B_out = 0;

                dome_active = 1;
                dome_out = 1;

                rasant_active = 1;
                rasant_out = 1;

                flex1_active = 1;
                flex1_out = 0;
                flex2_active = 1;
                flex2_out = 0;
                all_on_active = 0;
            }
            break;

        case 102:                       // Switch off all the sources
            all_off_active = 1;
            if(all_off_active == 1) {

                ringRGB_R_active = 0;
                ringRGB_R_out = 1;
                ringRGB_G_active = 0;
                ringRGB_G_out = 1;
                ringRGB_B_active = 0;
                ringRGB_B_out = 1;

                dome_active = 0;
                dome_out = 0;

                rasant_active = 0;
                rasant_out = 0;

                flex1_active = 0;
                flex1_out = 1;
                flex2_active = 0;
                flex2_out = 1;

                all_off_active = 1;
            }
            break;

        case 103:                       // Red light of the ring on
            if(ringRGB_R_active) {
                ringRGB_R_active = 0;
                ringRGB_R_out = 1;
                all_on_active = 0;
            } else {
                ringRGB_R_active = 1;
                ringRGB_R_out = 0;
            }
            break;

        case 104:                       // Green light of the ring on
            if(ringRGB_G_active) {
                ringRGB_G_active = 0;
                ringRGB_G_out = 1;
                all_on_active = 0;
            } else {
                ringRGB_G_active = 1;
                ringRGB_G_out = 0;
            }
            break;

        case 105:                       // Blue light of the ring on
            if(ringRGB_B_active) {
                ringRGB_B_active = 0;
                ringRGB_B_out = 1;
                all_on_active = 0;
            } else {
                ringRGB_B_active = 1;
                ringRGB_B_out = 0;
            }
            break;

        case 106:                       // Dome light on
            if(dome_active) {
```

```
                dome_active = 0;
                dome_out = 0;
                all_on_active = 0;
            } else {
                dome_active = 1;
                dome_out = 1;
            }
            break;

        case 107:                       // Allumer/Eteindre l'éclairage rasant selon l'état précédent
            if(rasant_active) {         // Si allumé
                rasant_active = 0;          // Changement de valeur pour allumer au prochain appui
                rasant_out = 0;             // Eteindre
                all_on_active = 0;          // Tous les éclairages ne sont pas allumés
            } else {                    // Si éteint
                rasant_active = 1;          // Changement de valeur pour éteindre au prochain appui
                rasant_out = 1;             // Allumer
            }
            break;

        case 115:                       // Placer le bras en position haute
            servo_mot_HB.pulsewidth_us(commande_HB(80));    // Correspond à un angle de 80°
            break;

        case 116:                       // Placer le bras en position basse
            servo_mot_HB.pulsewidth_us(commande_HB(0));     // Correspond à un angle de 0°
            break;

        case 117:                       // Déplier/Ranger le bras selon l'état précédent
            if(turn_active) {           // Si bras rangé
                turn_active = 0;            // Changement de valeur pour ranger le bras au prochain envoi de 117
                servo_mot_rot.pulsewidth_us(commande_rot(90)); // Dépliement du bras (rotation de 90°)
                }
            else {                      // Sinon (bras déployé)
                turn_active = 1;            // Changement de valeur pour déployer le bras au prochain envoi de 117
                rasant_active = 0;          // Eteindre l'éclairage
                rasant_out = 0;
                servo_mot_HB.pulsewidth_us(commande_HB(80));   // Replacer le bras en position haute
                wait(0.3f);
                servo_mot_rot.pulsewidth_us(commande_rot(0));  // Puis tourner de -90° pour ranger le bras
            }
            wait(0.2f);
            break;

        case 108:                       // Flex1 on/off
            if(flex1_active) {
                flex1_active = 0;
                flex1_out = 1;
                all_on_active = 0;
            } else {
                flex1_active = 1;
                flex1_out = 0;
            }
            break;

        case 109:                       // Flex2 on/off
            if(flex2_active) {
                flex2_active = 0;
                flex2_out = 1;
                all_on_active = 0;
            } else {
                flex2_active = 1;
                flex2_out = 0;
            }
            break;


        case 110:                               // Changer de direction
            if(conv_stop_active == 1) {         // Si le convoyeur est en marche
                if(conv_direction_active==1) {      // Et s'il est dans le sens horaire
                    conv_direction_active = 0;          // Changement de valeur pour changer de sens au prochain appui
                    conveyor_dir = CONV_DIR_RV;         // Changement de sens (RV = Reverse)
                } else {                            // Sinon (sens anti-horaire)
                    conv_direction_active = 1;          // Changement de valeur pour changer de sens au prochain appui
                    conveyor_dir = CONV_DIR_FW;         // Changement de sens (FW = Forward)
                }
            }
                break;

        case 111:                       // Arrêter/Démarrer le convoyeur selon l'état précédent
            if(conv_stop_active==1) {   // Si convoyeur en marche
                conv_stop_active = 0;       // Changement d'état pour le mettre en marche au prochain envoi de 111
                stopConveyor();             // Arrêter le convoyeur
            }
            else {                      // Sinon (convoyeur à l'arrêt)
                conv_stop_active = 1;       // Changement d'état pour l'arrêter au prochain envoi de 111
                if(conv_direction_active==1)  // Dépend du sens de rotation du convoyeur
                    moveConveyor(CONV_DIR_FW,periode);  // Réutiliser la vitesse avant arrêt
                else
                    moveConveyor(CONV_DIR_RV,periode);
            }
```

```
818             break;
819         default:
820             0;
821     }
822 }
823 
824 //__________Choisir le sens de rotation et la vitesse du convoyeur__________//
825 
826 void moveConveyor(char dir, int speed_us)   // Change la vitesse et le sens de rotation
827 {
828     conveyor_enable = 1;                // Activer le convoyeur
829     conveyor_speed.period_us(speed_us); // Changer la période du signal PWM pour changer la vitesse
830     conveyor_dir = dir;                 // Changer le sens de rotation (direction)
831     conveyor_speed.write(0.5);          // Rapport cyclique de 0.5 (arbitraire)
832 }
833 
834 //_____________________________Arrêter le convoyeur_____________________________//
835 
836 void stopConveyor(void)     // Arrête le convoyeur
837 {
838     conveyor_enable = 0;                // Désactiver le convoyeur
839     conveyor_speed.write(0);            // Rapport cyclique nul (signal de commande nul)
840 }
841 
842 //_____________________Contrôle des servomoteur du bras_____________________//
843 // Calculer la durée du front montant du signal PWM qui commande le servomoteur responsable de la rotation
844 int commande_rot(int angle){            // Faire tourner le bras de l'angle souhaité
845     return floor(8.9*angle+700);        // Loi Angle/Front montant déterminée expérimentalement
846     }
847 
848 // Calculer la durée du front montant du signal PWM qui commande le servomoteur responsable de la translation verticale
849 int commande_HB(int angle){             // Translation verticale associée à un angle entre 0° (bas) et 80° (haut)
850     return floor(-10.0*angle+2400);     // Loi Angle/Front montant déterminée expérimentalement
851     }
```

Annexe 3 : Code interface sur l'ordinateur

```
# AutoDarkRingTab.py

"""-------------------------------------------------------------------------------
|
| Vision Industrielle
|
-------------------------------------------------------------------------------
| Par l'équipe PIMS (Samuel Gerente, Hugo Lassiette, Flora Silberzan, Martin
Collignon, Oscar Boucher, Victoire de Saléon
| et par l'équipe ProTIS (Tong Zhu, Albane Lapras, Martin Collignon, Victoire de
Saléon)
|
-------------------------------------------------------------------------------
| La classe AutoDarkRingTab est appelée dans le main.py à la ligne 208 (dans la
classe MainWindow) par :
|     @Qtc.pyqtSlot()
|     def add_auto_dark_ring(self):
|         self.main_tab_widget.addTab(AutoDarkRingTab(self.video_thread,
self.ser), STR_AUTO_DARK_RING)
| cette fonction permet de rajouter un menu dans la fenêtre principale
|
-------------------------------------------------------------------------------
| ce menu permet de :
|     - visualiser l'aquisition vidéo si la caméra est allumée (les images de la
vidéo sont dans self.shown_cv_img)
|     - effectuer des traitements simple de l'image (ouverture fermeture flou
gaussien, pour une binarisation optimale)
|     - définir les paramètres de détection de contours des objets (aire minimale
et maximale des objets d'interêts)
|     - enregistrer une image de l'aquisition manuellement (soit dans le clipboard
soit dans un dossier)
| ET SURTOUT de :
|     - détecter les contours des objets
|     - déterminer leur centre et sa distance avec le centre de l'anneau
|     - si l'objet est au centre de l'anneau, baisser l'anneau et prendre une
photo enregistrée automatiquement dans un dossier, puis relever l'anneau
|
|
-------------------------------------------------------------------------------"""
|
| import numpy as np
|
| #permet de créer des interfaces en python :
| from PyQt5 import QtWidgets as Qtw
| from PyQt5 import QtCore as Qtc
|
| #fonctions de bases du traitement de l'image et de manipulation d'images :
| import cv2
|
| #fonctions programmées pendant les semaines PIMS
| from Utils import convert_cv_to_qpixmap, Timer, h_line, is_gray
| from PreProcessingFunction import func_contours_list, gray_to_binary
| from FormDetectionFunction import is_outside
|
| #dictionnaire de chaines de caractères fait pendant les semaines PIMS
| from Dictionary_EN import *
|
|
| class AutoDarkRingTab(Qtw.QMainWindow):
|     def __init__(self, video_thread, ser):
|         super().__init__()
|         self.nb=0 # numérotation des photos
|         self.ser = ser # pour la conection usb
|
|         self.raw_cv_img = None
|         self.shown_cv_img = None
```

```
052|          self.calculating = False # Bouton de calcul en mode off
053|
054|          self.get_video_thread = video_thread
055|
self.get_video_thread.new_cv_img_signal.connect(self.update_video_qlabel)
056|          self.get_video_thread.stopped_signal.connect(self.video_stopped)
057|
058|          self.video_qlabel = Qtw.QLabel()
059|          self.setCentralWidget(self.video_qlabel)
060|
061|          self.setting_dock = Qtw.QDockWidget('Setting', self)
062|          setting_widget = Qtw.QWidget()
063|
064|          self.calculate_btn = Qtw.QPushButton(STR_START_CALCULATION)
065|          self.calculate_btn.clicked.connect(self.start_calculation)
066|          self.calculate_btn.setEnabled(False)
067|
068|          self.fps_qlabel = Qtw.QLabel()
069|
070|          """Pre-processing Buttons"""
071|
072|          self.show_binary_checkbox = Qtw.QCheckBox(STR_SHOW_BINARY_IMAGE)
073|          self.show_binary_checkbox.setChecked(False)
074|
075|          self.opening_checkbox = Qtw.QCheckBox(STR_OPENING)
076|          self.opening_checkbox.setChecked(False)
077|
078|          self.closing_checkbox = Qtw.QCheckBox(STR_CLOSING)
079|          self.closing_checkbox.setChecked(False)
080|
081|          self.blur_checkbox = Qtw.QCheckBox(STR_GAUSSIAN_BLURRING)
082|          self.blur_checkbox.setChecked(False)
083|
084|          self.threshold_spinbox = Qtw.QSpinBox()
085|          self.threshold_spinbox.setRange(-1, 255)  # -1 value is for auto
threshold selection
086|          self.threshold_spinbox.setValue(80) #valeur du seuil fixée à 80
087|
088|          self.min_area_spinbox = Qtw.QSpinBox()
089|          self.min_area_spinbox.setRange(0, 1000000000)
090|          self.min_area_spinbox.setSingleStep(50)
091|          self.min_area_spinbox.setValue(1000)
092|
093|          self.max_area_spinbox = Qtw.QSpinBox()
094|          self.max_area_spinbox.setRange(0, 1000000000)
095|          self.max_area_spinbox.setSingleStep(50)
096|          self.max_area_spinbox.setValue(90000)
097|
098|          """Screenshot Buttons"""
099|
100|          self.screenshot_btn = Qtw.QPushButton(STR_TAKE_SCREENSHOT)
101|          self.screenshot_btn.clicked.connect(self.save_screenshot)
102|          self.screenshot_btn.setEnabled(False)
103|          self.clipboard_btn = Qtw.QPushButton(STR_COPY_TO_CLIPBOARD)
104|          self.clipboard_btn.clicked.connect(self.clipboard_screenshot)
105|          self.clipboard_btn.setEnabled(False)
106|
107|          """Layout"""
108|          #on ajoute tous les éléments au layout
109|
110|          setting_form_layout = Qtw.QFormLayout()
111|          setting_form_layout.addRow(self.calculate_btn) # ajout du bouton pour le
calcul de la position des objets
112|          setting_form_layout.addRow(h_line()) # petite ligne horizontale pour que
ca soit joli
113|          setting_form_layout.addRow(Qtw.QLabel(STR_FPS), self.fps_qlabel) #chaîne
de caractères pour indiquer le nombre d'images par secondes
```

```
114|         setting_form_layout.addRow(h_line()) # petite ligne horizontale pour que
ca soit joli
115|         setting_form_layout.addRow(self.show_binary_checkbox) # petite checkbox
pour montrer l'image binaire
116|         setting_form_layout.addRow(self.opening_checkbox) # petite checkbox pour
effectuer une ouverture
117|         setting_form_layout.addRow(self.closing_checkbox) # petite checkbox pour
effectuer une fermeture
118|         setting_form_layout.addRow(self.blur_checkbox) # petite checkbox pour
effectuer une fermeture
119|         setting_form_layout.addRow(STR_THRESHOLD, self.threshold_spinbox) #
petite spinbox pour choisir la valeur du seuil pour le seuillage
120|         setting_form_layout.addRow(STR_MIN_SURFACE, self.min_area_spinbox) #
petite spinsbox pour choisir la taille minimale d'un objet
121|         setting_form_layout.addRow(STR_MIN_SURFACE, self.max_area_spinbox)
122|         setting_form_layout.addRow(h_line())
123|         setting_form_layout.addRow(self.screenshot_btn, self.clipboard_btn)
124|
125|         setting_widget.setLayout(setting_form_layout)
126|
127|         self.setting_dock.setWidget(setting_widget)
128|         self.addDockWidget(Qtc.Qt.LeftDockWidgetArea, self.setting_dock)
129|         self.timer = Timer()
130|         self.timer.start()
131|
132|     def video_stopped(self):
133|         self.video_qlabel.setText(STR_PLS_LAUNCH_ACQUISITION)
134|         self.shown_cv_img = None
135|         self.fps_qlabel.setText('')
136|         self.calculating = False
137|         self.calculate_btn.setEnabled(False)
138|         self.calculate_btn.setText(STR_START_CALCULATION)
139|         self.screenshot_btn.setEnabled(False)
140|         self.clipboard_btn.setEnabled(False)
141|
142|     @Qtc.pyqtSlot()
143|     def save_screenshot(self): # permet d'ouvrir une fenêtre pour enregistrer le
screenshot
144|         filename = Qtw.QFileDialog.getSaveFileName(self, STR_SAVE_IMAGE, '',
STR_IMAGE_FORMAT)
145|         print(filename[0])
146|         if filename[0] != "":
147|             cv2.imwrite(filename[0], self.shown_cv_img)
148|
149|     @Qtc.pyqtSlot()
150|     def clipboard_screenshot(self): # permet de copier l'image dans le clipboard
151|
Qtw.QApplication.clipboard().setPixmap(convert_cv_to_qpixmap(self.shown_cv_img,
None))
152|
153|     @Qtc.pyqtSlot()
154|     def start_calculation(self):
155|         if not self.calculating:
156|             self.calculate_btn.setText(STR_STOP_CALCULATION)
157|             self.calculating = True
158|         else:
159|             self.calculate_btn.setText(STR_START_CALCULATION)
160|             self.calculating = False
161|             self.fps_qlabel.setText('')
162|
163|     @Qtc.pyqtSlot(np.ndarray)
164|     def update_video_qlabel(self, cv_img): # met à jour l'image avec la nouvelle
image de la caméra
165|         if is_gray(cv_img):
166|             self.calculate_btn.setEnabled(True)
167|             self.screenshot_btn.setEnabled(True)
168|             self.clipboard_btn.setEnabled(True)
```