HAUTE ÉCOLE DE NAMUR-LIEGE-LUXEMBOURG

DÉPARTEMENT INGÉNIEUR INDUSTRIEL

PIERRARD - VIRTON

Système intelligent

Rapport

Niryo Vision

Année académique
2018-2019
Master 1 - finalité automatisation

Gillet Niels
Houba Guillaume
Hartman Nicolas

Table des matières

1. Tables des références :

1.1. Tables des figures

1.2. Tables des tableaux

2. Cahier des charges :

Enoncé travail UE Systèmes intelligents :

Le travail sera évolué tout au long des séances de cours de Microcontrôleurs et de Systèmes intelligents à chaque étape validée.

Partie 1 : deadline 26/02/19

Avant de réaliser votre travail, une analyse devra être réalisée (sous forme de document ou de présentation). Cette analyse reprendra :

- Une description du projet
- Une description du matériel nécessaire (indiquer le code RS pour la commande)
- Un diagramme du projet
- La répartition des tâches au sein du groupe
- Une time line avec des étapes de validations
- Un état de l'art de l'existant (code, page-web…), critiques et plan de réutilisation

Partie 2 :

Après validation du projet par l'équipe enseignante, il pourra débuter.

Il sera évaluer à la fin de chaque étape de la time line.

A chaque étape, il faudra remettre sur Moodle :

- Un paragraphe reprenant l'avancement du projet (time line)
- Le répertoire contenant les codes (avec commentaires et documentations de chaque fonction **en anglais**)

Partie 3 :

Une présentation (10 min) et défense (10-20 min) aura lieu lors de la séance de cours du **20 mai 2019**.

Le répertoire contenant les codes (avec commentaires et documentations de chaque fonction **en anglais**) devront être déposés sur Moodle au plus tard le **15/05/19**.

<table>
<tr><th colspan="3">Pondération du travail</th></tr>
<tr><td colspan="2">Partie 1 : Analyse</td><td>20%</td></tr>
<tr><td>Partie 2 : Réalisation</td><td>• Installer et utiliser git pour le traçage des documents et du code !
• Fonctionnement du code
• Précision du résultat
• Optimisation du matériel
• Application intermédiaire
• Application finale</td><td>80%</td></tr>
<tr><td rowspan="2">Partie 3 : Présentation</td><td>Présentation</td><td rowspan="2">Examen : 50%</td></tr>
<tr><td>Défense</td></tr>
</table>

Figure 1: Cahier des charges

3. Énoncé de rédaction du rapport :

Système Intelligent – M1 auto Henallux

Tableau d'évaluation

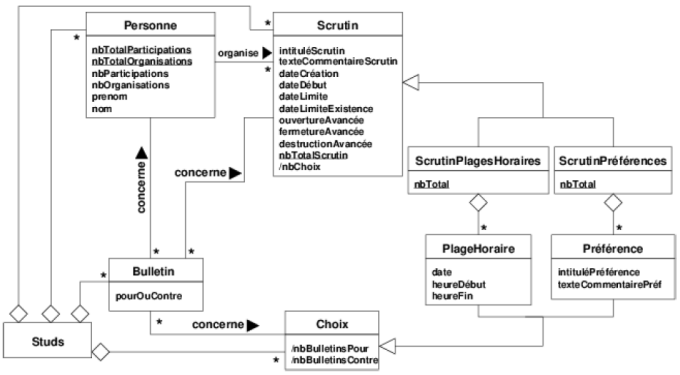
Analyse (déjà notée)	20%
Rapport	30%
Présentation	20%
Défense groupe	15%
Défense individuelle	15%

Rapport

Le rapport comportera :

1) Introduction : explication du projet ainsi que son but.
2) Le tableau des composants (lien + prix) et le schéma explicatif de la fonctionnalité de chaque composant.
3) Time line final.
4) Partie 1 : Arduino.
5) Partie 2 : Rasbperry + Reconnaissance.
6) Conclusion : Limites du projet et possibilités d'amélioration

Dans les 2 parties (point 4 et 5 ci-dessus) du rapport aucun code n'est attendu, mais le référencement vers le git est obligatoire. L'explication sera réalisée via des diagrammes de classe ou de fonctionnalité (voir exemple ci-dessous). Ces parties contiendront également un état de l'art (lien) pour critiquer les sources software utilisées.



Présentation orale

La présentation orale reprendra les points forts de votre projet ainsi que le résumé de votre rapport. (Ne pas reprendre les parties time line et composants). Cette présentation finira par une démo vidéo (2-3 min).

- Support pptx
- Durée :
  - 10 min présentation
  - 10 min défense groupe
  - 5 min/étudiant interrogation individuelle

Figure 2: Modalité d'évaluation

4. Diagramme de Gantt (Time line) :

N°	Mode Tâche	Nom de la tâche	Durée	Début	Fin	Prédécesseurs
0		**NiryoVision**	**54,13 jrs?**	**Mar 26/02/19**	**Mer 22/05/19**	
1		**1 Découverte du matériel**	**14,13 jrs**	**Mar 26/02/19**	**Mar 19/03/19**	
2		1.1 Liaison du Raspberry avec la caméra	9,63 jrs	Mar 26/02/19	Mar 12/03/19	
3		1.2 Commande du robot avec raspberry	6,13 jrs	Lun 11/03/19	Mar 19/03/19	
4		**2 Adaptation à notre projet**	**18,5 jrs**	**Mer 13/03/19**	**Mer 10/04/19**	
5		2.1 Reconnaissance des objets avec openCV	18,38 jrs	Mer 13/03/19	Mer 10/04/19	2
6		2.2 Déplacement du robot en fonction de l'environnement	13,88 jrs	Mer 20/03/19	Mer 10/04/19	3
7		**3 Finalisation du programme**	**26,13 jrs?**	**Jeu 11/04/19**	**Mer 22/05/19**	**4**
8		3.1 Liaison des deux parties de programme	6,88 jrs?	Jeu 11/04/19	Lun 22/04/19	5;6
9		3.2 Création d'un cycle de fonctionnement	19,13 jrs?	Mar 23/04/19	Mer 22/05/19	8
10		3.3 Débogage et finalisation	19,13 jrs?	Mar 23/04/19	Mer 22/05/19	8
11		3.4 Création du multithreading	7 jrs	Lun 6/05/19	Mer 15/05/19	

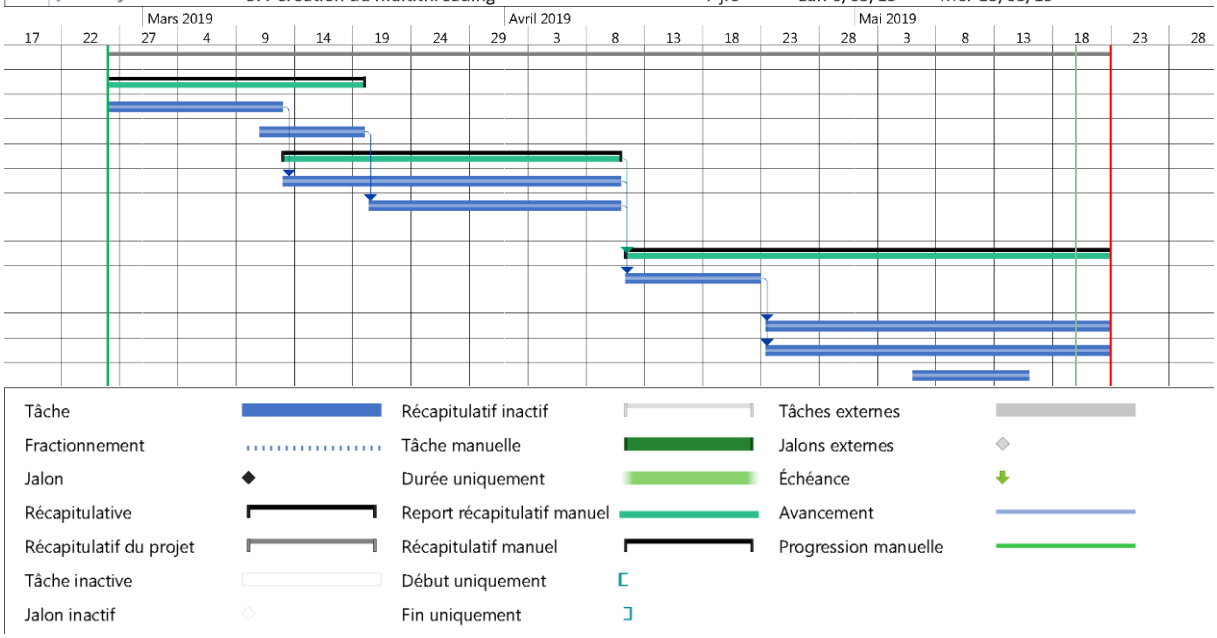


Figure 3: Diagramme de Gantt

5. Introduction :

Dans la cadre du cours de système intelligent, nous avons décidé de travailler sur un robot 6 axes. L'objectif est d'automatiser le robot pour qu'il soit capable de trier des pièces en fonction de leurs formes. Les formes sont carrées, rondes, triangulaires et en étoiles. Elles ont été imprimées en 3D. Nous avons imposé de reconnaitre uniquement les pièces noires sur fond d'une table blanche et uniquement les quatre formes présentes dans la Figure 4.

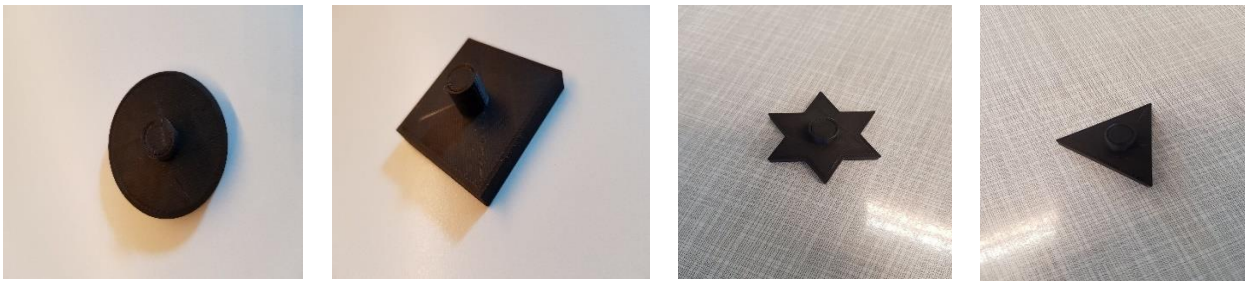


Figure 4: Les formes

Pour effectuer le tri, nous avons ajouté un Raspberry3 et une caméra Pi au robot.

Le Raspberry est utilisé pour traiter l'image filmée par la caméra Pi. Celle-ci est accrochée sur un des axes du robot. La vidéo en temps réel de la caméra est affichée sur un écran LCD.

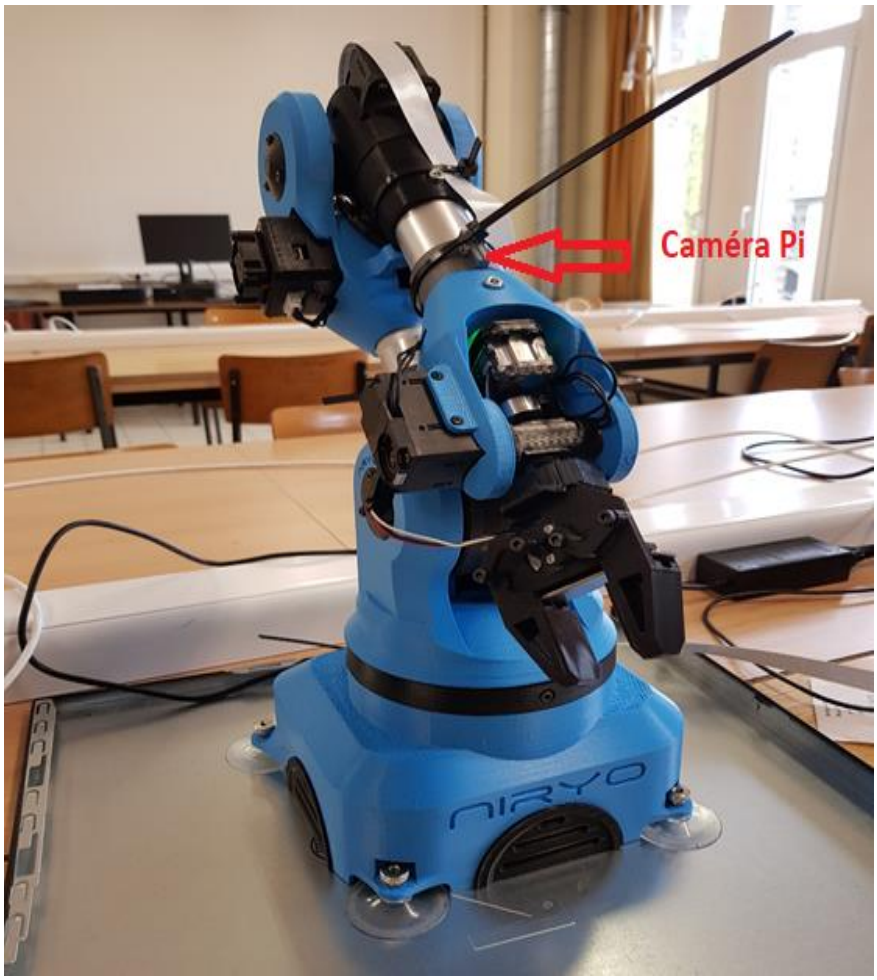


Figure 5: Robot Niryo One

Le robot est doté de deux modes : manuel et automatique.

Le mode automatique permet au robot de faire un « cycle ». Dans ce mode, le robot positionne la caméra au-dessus de chaque pièce pour permettre au Raspberry de l'analyser. Une fois l'analyse terminée, la pince du robot prend la pièce et la dépose dans la boîte prévue à cet effet voir la Figure 6. Une fois le cycle terminé, le robot se remet en mode « pause » illustré à la Figure 5.

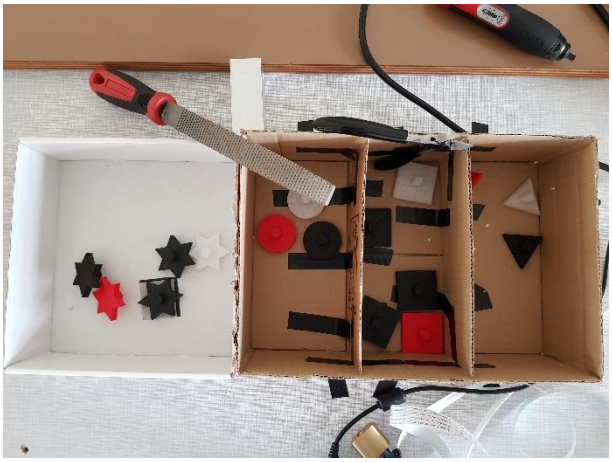


Figure 6: Le réceptacle à pièce

Le mode manuel permet à l'utilisateur de positionner le bras à des positions prédéfinies. Il existe 3 positions possibles pour chaque pièce : position haut, position bas et position caméra. De plus, il est possible d'ouvrir ou de fermer la pince dans ce mode.

La position haute, Figure 7, permet de se placer juste au-dessus de la pièce pour pouvoir correctement les placer au début de l'opération, la position basse, Figure 8, quant à elle nous permettra de prendre la pièce et la position. On a choisi la position caméra (Figure 9) de manière optimale afin que la caméra ait une position de vue adaptée pour la reconnaissance de forme.

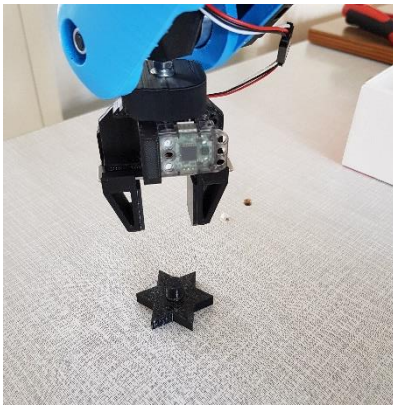


Figure 7: Position haute

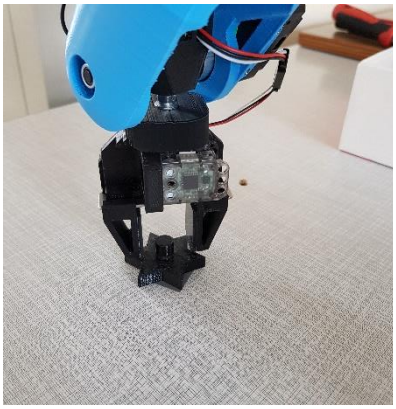


Figure 8: Position basse

Figure 9: Position caméra

6. Liste des composants :

Voici un tableau récapitulatif qui reprend les différents composants qui constituent le projet : la caméra est utilisée pour filmer et afficher le résultat sur l'écran LCD. Le Raspberry sert à traiter les informations et piloter le Robot Niryo One. Ce Robot est quant à lui utilisé pour prendre les différentes pièces et les déposer aux endroits prédéfinis. Le câble est utilisé pour transférer les informations de la caméra au Raspberry.

Nom du composant	Prix	Lien
Robot Niryo One	1799€	https://niryo.com/fr/
Caméra Pi	14,95€	https://www.gotronic.fr/cat-cameras-1646.htm
Écran LCD	49.95€	https://www.gotronic.fr/art-ecran-tactile-5-lcd5-pour-raspberry-pi-25666.htm
Raspberry 3	40€	https://www.gotronic.fr/cat-cartes-raspberry-1564.htm
Câble FPC 2m	7 €	https://www.gotronic.fr/art-nappe-2-m-pour-camera-pour-raspberry-pi-27963.htm
Total	**1910,9 €**	

Tableau 1: Liste des composants

7. Multithreading :

L'exécution du code a dû se faire sur trois threads en même temps pour pallier les ralentissements de la capture vidéo et de l'affichage durant le déplacement du bras robotique du robot Niryo One. Le code fut pensé développer comme illustré à la Figure 10.

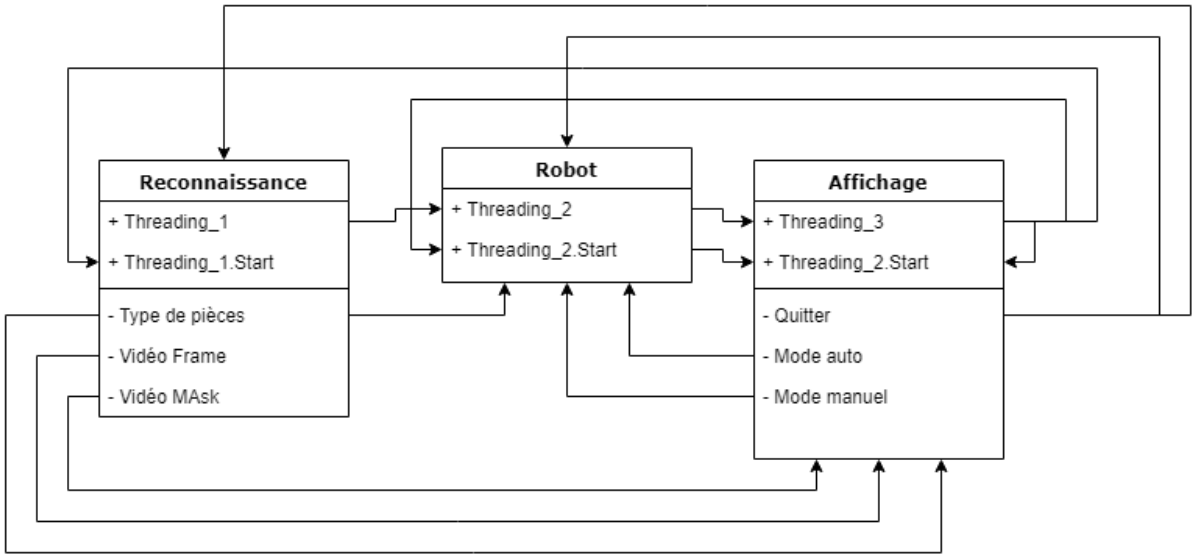


Figure 10: Représentation du multithreading et valeurs partagées

Nous disposons ici de trois threads différents, un qui va traiter le robot c'est-à-dire le déplacement du bras robotique de celui-ci en mode automatique, le déplacement du bras à chaque position que ce soit la position haute, basse, caméra …. Un deuxième thread reconnaissance qui va capturer la vidéo renvoyée par la caméra raspberryPi, en extraire le masque, qui lui aura été défini préalablement pour ne reconnaitre que les pièces noires, et va reconnaitre le contour pour ensuite le greffer à l'image principale. Cette reconnaissance de contour/forme sera envoyée au robot qui sera de quel type de pièce il s'agit. Et pour finir un dernier thread qui va gérer tout ce qui est affichage graphique, c'est-à-dire la reprise des captures vidéo avec les ajouts d'openCV (contour et masque) qui sont expliquées au chapitre 8.1 et qui sont renvoyées depuis le thread reconnaissance.

Les threads sont créés et lancés dans l'ordre expliqué à la Figure 11, cet ordre n'a pas été choisi aléatoirement, s'il n'est pas respecté le thread affichage va tenter d'afficher ou de communiquer avec des objets, méthodes variables qui ne sont pas encore présentes.

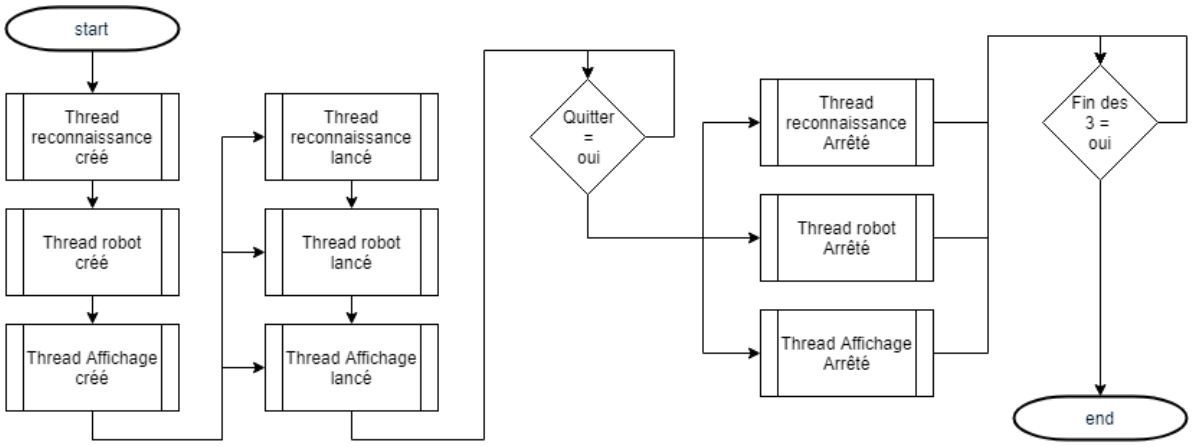


Figure 11: Ordre de lancement

8. Raspberry et reconnaissance :

Deux techniques ont été utilisées pour la reconnaissance des formes. La première est une reconnaissance de forme sans « machine learning » et la deuxième utilise le « machine learning » avec des descripteurs (**inachevée et en bonus**). La première méthode fut utilisée, car elle est suffisamment simple et puissante pour le type d'application ici présente, qui a été défini en corrélation avec le cahier de charge et dans l'introduction au chapitre 5.

Il existe plusieurs techniques pour analyser une image et en extraire les objets ou tout autre élément il existe une multitude de méthodes que nous allons citer brièvement dans le Tableau 2 et sans rentrer dans les détails.

Nom
inRange (Technique utilisée dans notre cas)
Canny
Compare

Tableau 2: Les différentes méthodes d'extractions d'objets

8.1. Technique 1 :

Le but de cette technique est de créer un masque. Ce qui veut dire que le Raspberry analyse tous les pixels de la caméra. Si la couleur du pixel se situe dans la plage voulue représentée à la

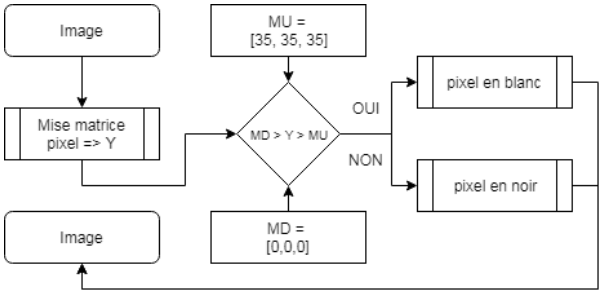


Figure 12 et à la Figure 13, il le colore en blanc. À l'inverse, le Raspberry le colore en noir. Voici le résultat pour un carré qui est illustré à la Figure 14.

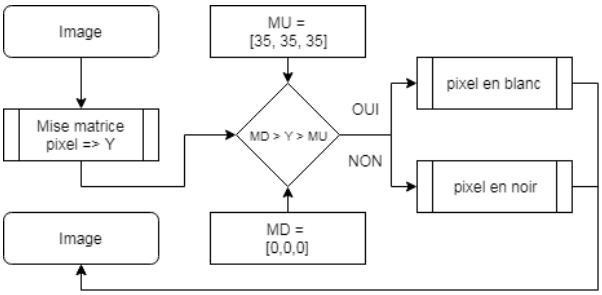


Figure 12: Ligne de code des masques

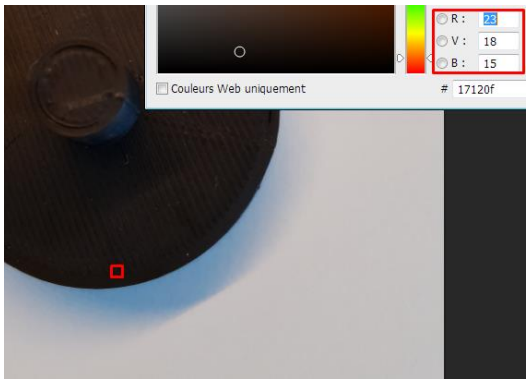


Figure 13: Image indiquant les valeurs RGB de l'image

Figure 14: Masque de la première technique

L'utilisation du masque oblige de travailler avec des pièces dont la couleur se situe dans une certaine plage et sur une table de couleur complètement différente. Dans notre cas, nous avons utilisé des pièces noires sur une table blanche pour cette raison.

Une fois le masque crée, il faut tracer le contour du masque. Python est capable de le faire très facilement en analysant les pixels. Si deux pixels l'un à côté de l'autre sont différents, cela veut dire que le pixel se trouve sur le contour de la forme. Une fois tous les pixels analysés, le Raspberry reconnaît le contour de la forme. Voici le résultat du contour affiché sur la forme. On peut voir sur la Figure 15 que le résultat est précis.

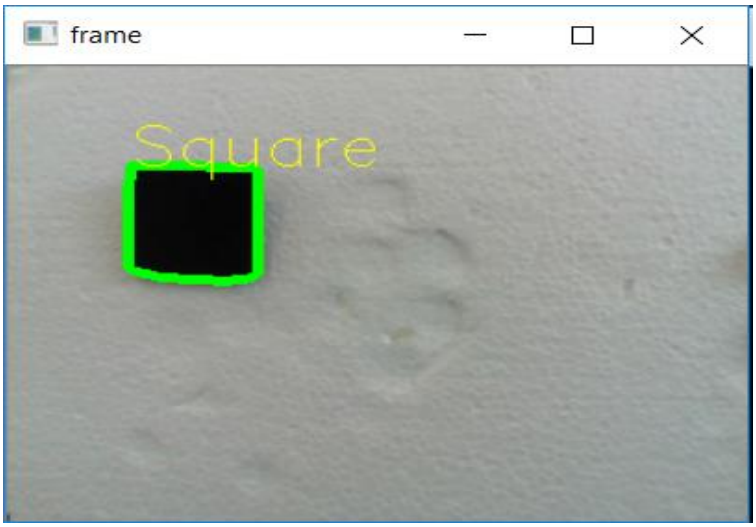


Figure 15 : Résultat de la première technique

La dernière étape est de compter le nombre de changements d'angle significatifs. En fonction du nombre de changements d'angle, il en déduit la forme. Dans le cas du carré, il faut d'abord approximer le contour pour qu'il soit constitué de lignes droites. Le raspberry écrit le résultat directement sur l'affichage vidéo.

Cette technique est contraignante pour une raison : le multithreading qui a été abordé au chapitre 7.

Voici notre diagramme de classe pour cette technique :

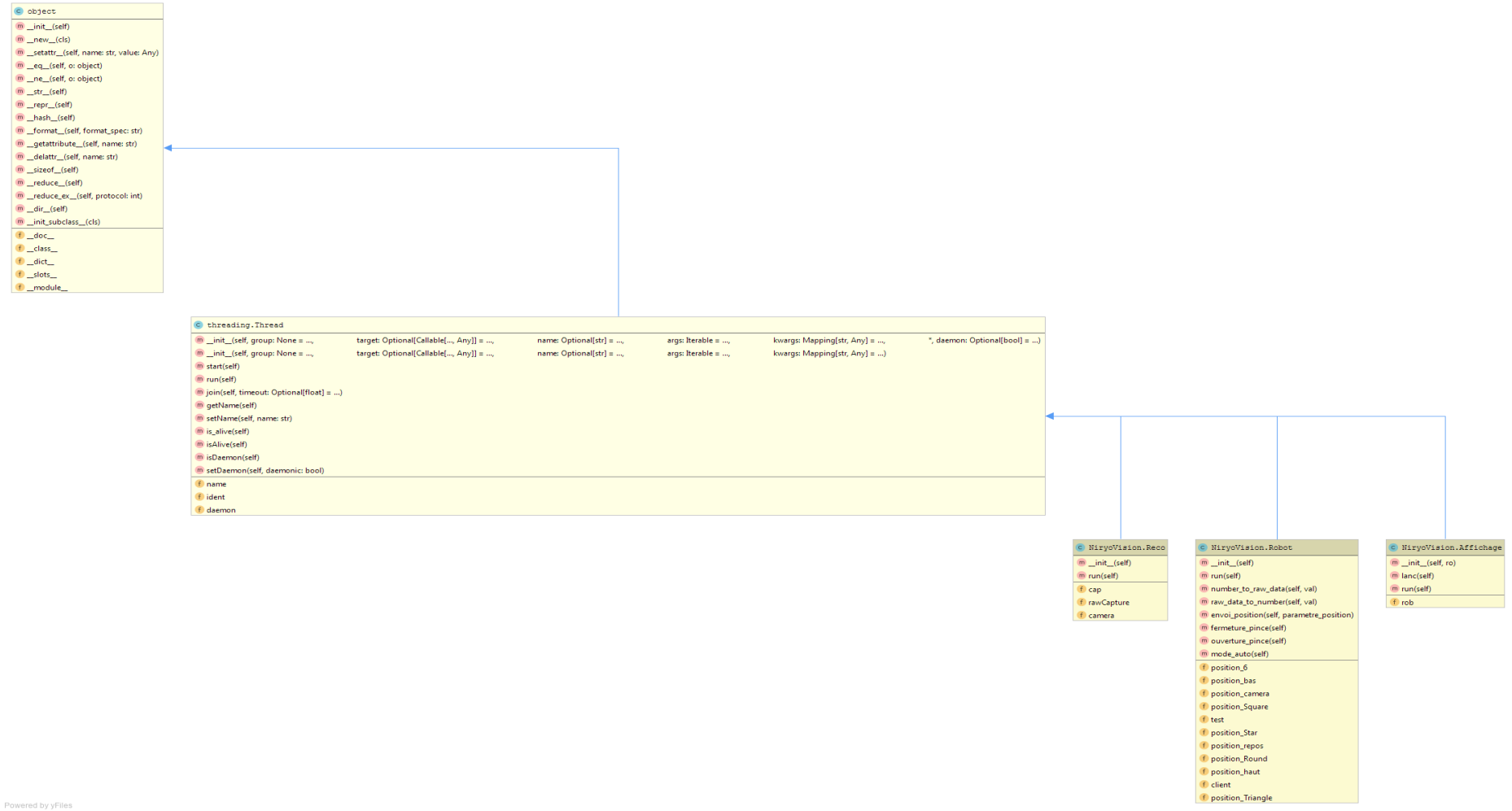


Figure 16: Diagramme de classe de la première technique

8.2. Technique 2 :

Cette technique utilise une des méthodes présentes dans le « machine learning », ici nous avons utilisé kNN (k-Nearest Neighbors) qui est de type supervisé cette à dire qu'il va comparer sons modèle et ses résultats avec une base de données contenant les résultats réels et corrects.

Premièrement, nous avons créé une base de données avec des photos des différentes formes. C'est avec celle-ci que le programme va apprendre.

Pour que le kNN soit précis, il faut traiter les données. Dans notre cas, nous avons classé les formes dans 4 listes différentes {Rond, Carré, Triangle, Étoile}. Ensuite, nous avons utilisé deux descripteurs différents : SIFT et ORB. Ces descripteurs vont permettre de trouver les points importants de chaque image. Voici un exemple avec le descripteur SIFT présenté aux Figure 17 et Figure 18:

Figure 17: Image brute

Figure 18: Image brute + Kp

Une fois les « keypoints » (kp) trouvés, ils sont transformés en vecteurs. Étant donné que nous travaillons avec « x » kp par images, nous devons en faire un vecteur de dimension [1 ; x] par image, une distance euclidienne est effectuée sur le vecteur des « x » images de chaque classe. Ensuite, chaque distance est labellisée pour qu'elle soit reliée à sa forme/ classe. Voici un schéma récapitulatif.

X_label	Y
distance 1	1
distance 2	1
.	1
.	1
.	1
distance x	1

X_label	Y
distance 1	2
distance 2	2
.	2
.	2
.	2
distance x	2

X_label	Y
distance 1	3
distance 2	3
.	3
.	3
.	3
distance x	3

X_label	Y
distance 1	4
distance 2	4
.	4
.	4
.	4
distance x	4

1	rond
2	carré
3	triangle
4	étoile

Figure 19: représentation des Ylabel et Xlabel

La troisième étape consiste à faire différences entre toutes ces distances. Le résultat de cette différence doit être différent en fonction du y_label. Par exemple, la différence entre ronds ronds sera plus petite et différente que la différence entre ronds carrés. Pour savoir si les différences trouvées sont représentatives, nous avons tracé un graphique pour les ronds et les carrés. Voici le résultat :

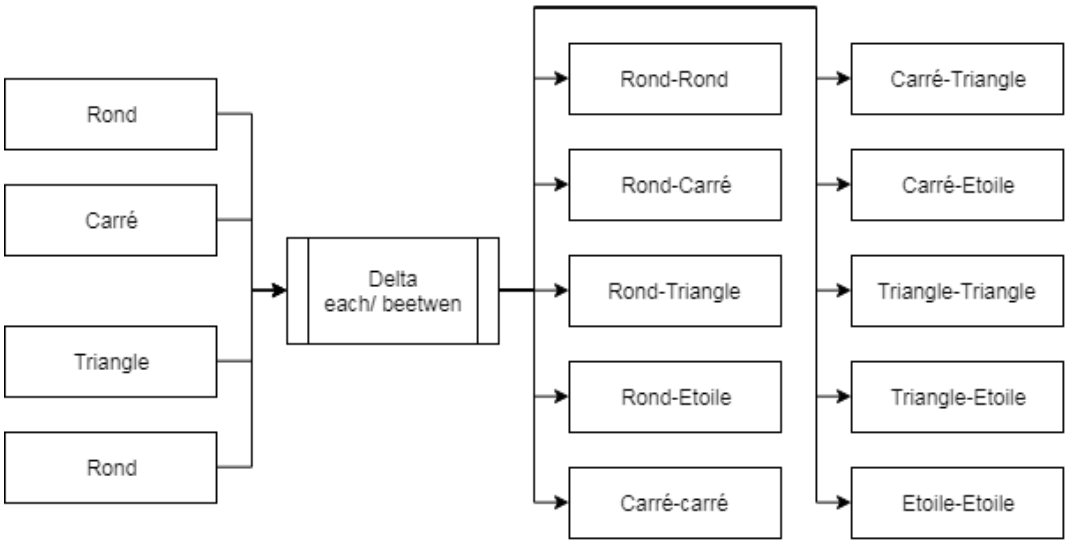


Figure 20: Illustration de la méthode des distances

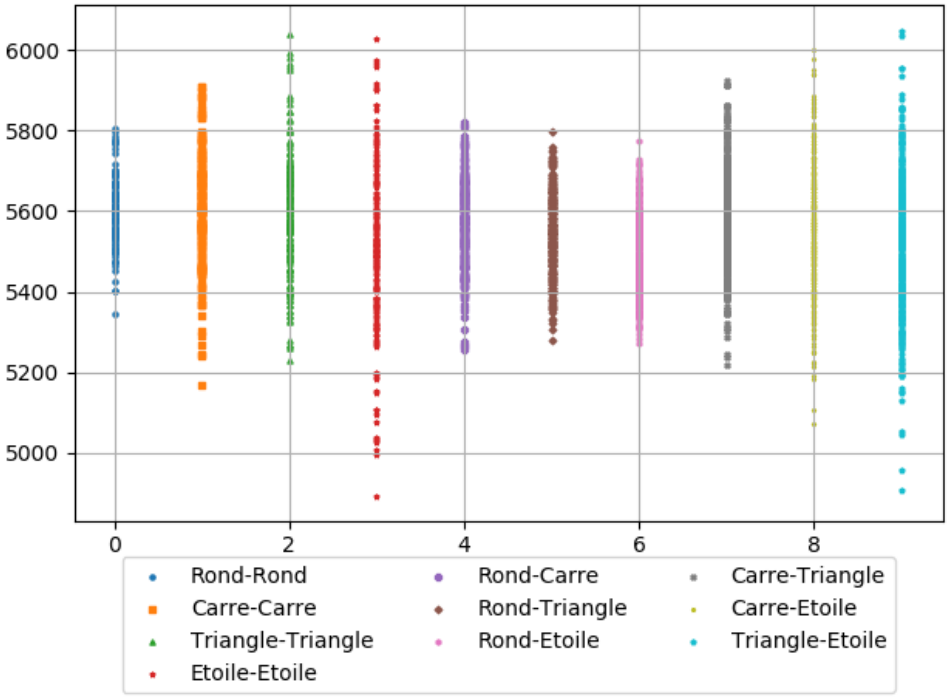


Figure 21: Graphique du résultat

On peut voir ici le problème sur lequel nous sommes tombés. Les différences trouvées sont presque toutes les mêmes, quelles que soient les formes. C'est à cause de cela que nous n'avons pas réussi à finir cette technique.

Voici notre diagramme de classe pour cette technique :

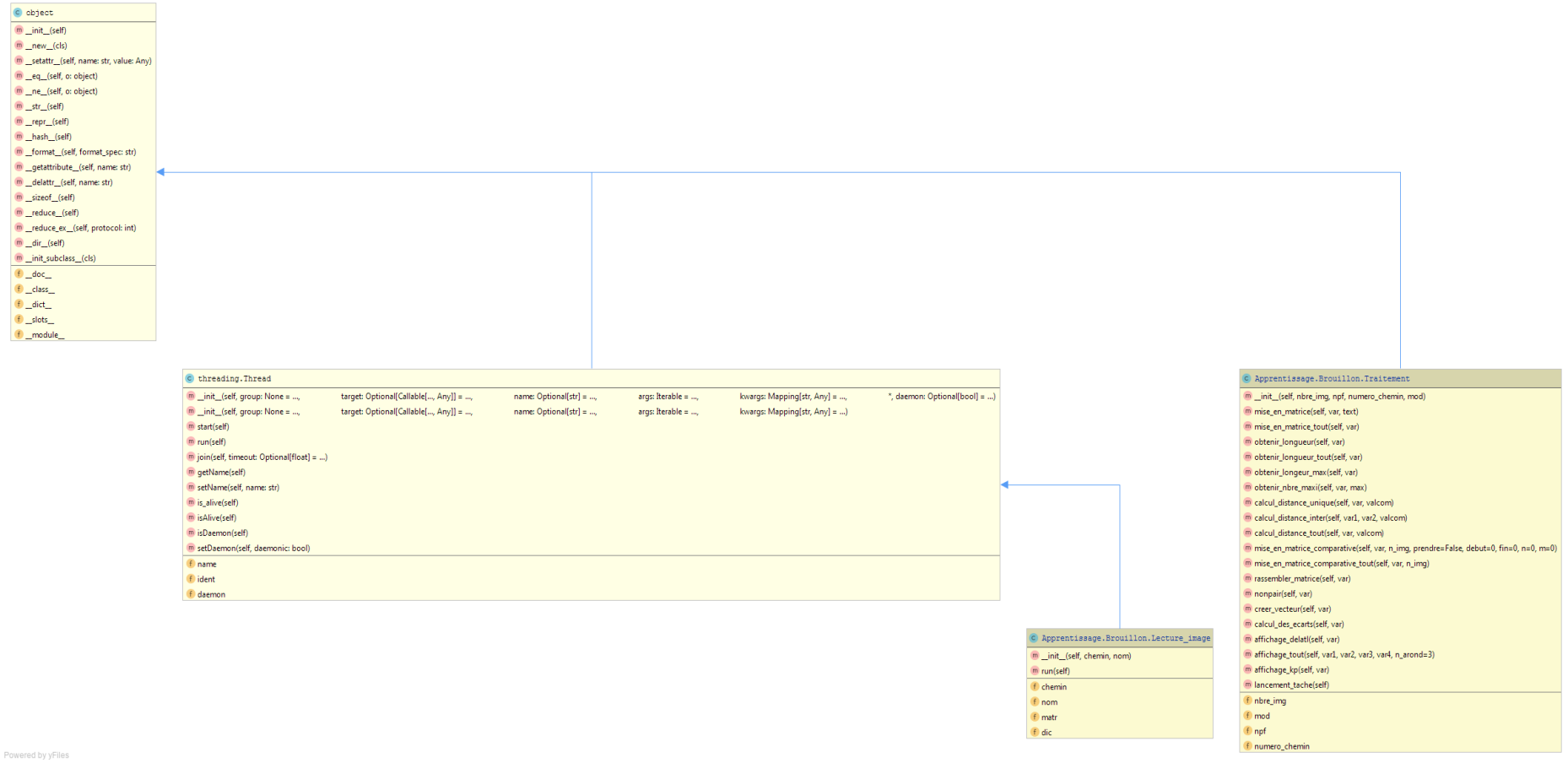


Figure 22: Diagramme de classe de la deuxième technique

9. Le robot :

Le robot Niryo one est un robot 6 axes pilotés en interne par un Raspberry sous le système ROS. Ce système permet de simplifier grandement la programmation, en effet plutôt que de gérer les moteurs manuellement, nous envoyons simplement les positions voulues et le robot s'y rend en déplaçant au mieux ses derniers avec les vitesses adaptées.

Plusieurs protocoles sont disponibles. Nous utiliserons le Modbus TCP/IP qui semble être le plus adapté dans notre cas. Ce mode demande de démarrer un programme sur le serveur (ici le robot) pour envoyer des commandes depuis le client (notre Raspberry avec écran). En choisissant le bon port sur le client avec l'adresse IP du robot, le transfert de commandes s'effectue sans aucun souci.

Les moyens de communiquer avec le robot sont détaillés sur cette page. La documentation des commandes pour le protocole Modbus est détaillée sur cette page, les principales utilisées sont montrées sur la Figure 23.

Holding Register

Each address contains a 16bit value.

READ/WRITE (the stored values correspond to the last given command, not the current robot state)

Accepted Modbus functions :

- 0x03: READ_HOLDING_REGISTERS
- 0x06: WRITE_SINGLE_REGISTER

Address	Description
0-5	Joints (mrad)
10-12	Position x,y,z (mm)
13-15	Orientation roll, pitch, yaw (mrad)
100	Send Joint Move command with stored joints
101	Send Pose Move command with stored position and orientation
110	Stop current command execution
150	Is executing command flag
151	Last command result*
200-299	Can be used to store your own variables
300	Learning Mode (On = 1, Off = 0)
301	Joystick Enabled (On = 1, Off = 0)
310	Request new calibration
311	Start auto calibration
312	Start manual calibration
401	Gripper open speed (1-5)
402	Gripper close speed (1-5)
500	Select tool from given id **
510	Open gripper with given id
511	Close gripper with given id
512	Pull air vacuum pump from given id
513	Push air vacuum pump from given id

*The "Last command result" gives you more information about the last executed command :

- 0 : no result yet
- 1 : success
- 2 : command was rejected (invalid params, ...)
- 3 : command was aborted
- 4 : command was canceled
- 5 : command had an unexpected error
- 6 : command timeout
- 7 : internal error

** Select tool from id : you can find the tools ids here. Send id "0" to detach current tool.

Figure 23 Commandes ModBus pour le robot Niryo

9.1. Fixation de la caméra sur le robot :

La caméra Raspberry pi doit être fixée sur le bras du robot de manière optimale pour qu'elle ne bouge pas durant les mouvements du robot. À cet effet, nous avons modélisé un support pour l'imprimer en 3D comme présenté sur la Figure 24. Ce support est composé de deux parties reliées par une charnière d'un côté à gauche de la Figure 24 et d'une vis de serrage dans un insert fileté pour fixer fermement la caméra au bras à droite de la Figure 24.
La caméra est ensuite fixée par les trous de serrage sur le dessus présent sur la Figure 24.

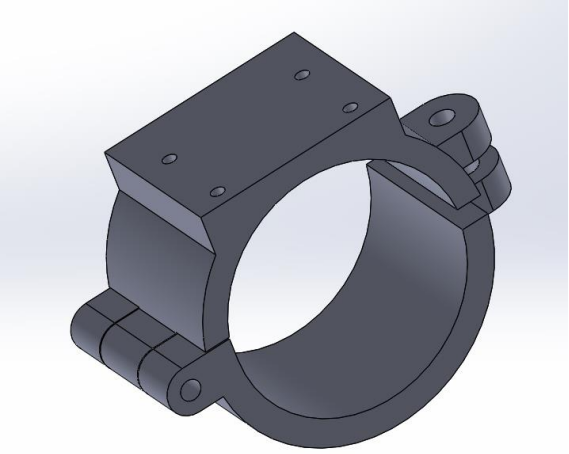


Figure 24: Modélisation du support caméra

9.2. Interface graphique :

Afin de gérer facilement les différentes fonctionnalités de notre programme, nous avons créé une interface graphique contenant la vision de la caméra, le lancement du mode automatique et une page dédiée au mode manuel permettant de positionner le robot à toutes les positions voulues.

Cette dernière est réalisée avec la bibliothèque TKINTER simplifiant grandement l'intégration de toutes les fonctionnalités voulues. Elle prend naturellement tout l'écran 5 pouces de 800 par 480 et permet de sélectionner aisément les boutons voulus avec le stylet tactile.

Voici le menu principal et le menu du mode manuel (Figure 25 & Figure 26)

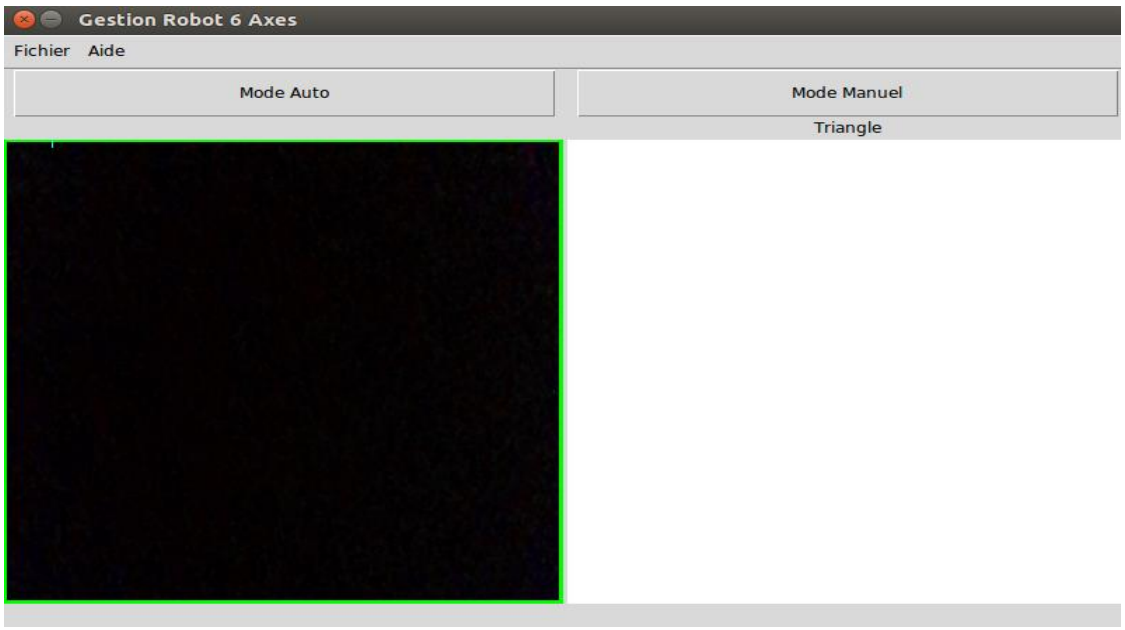


Figure 25: Menu principal

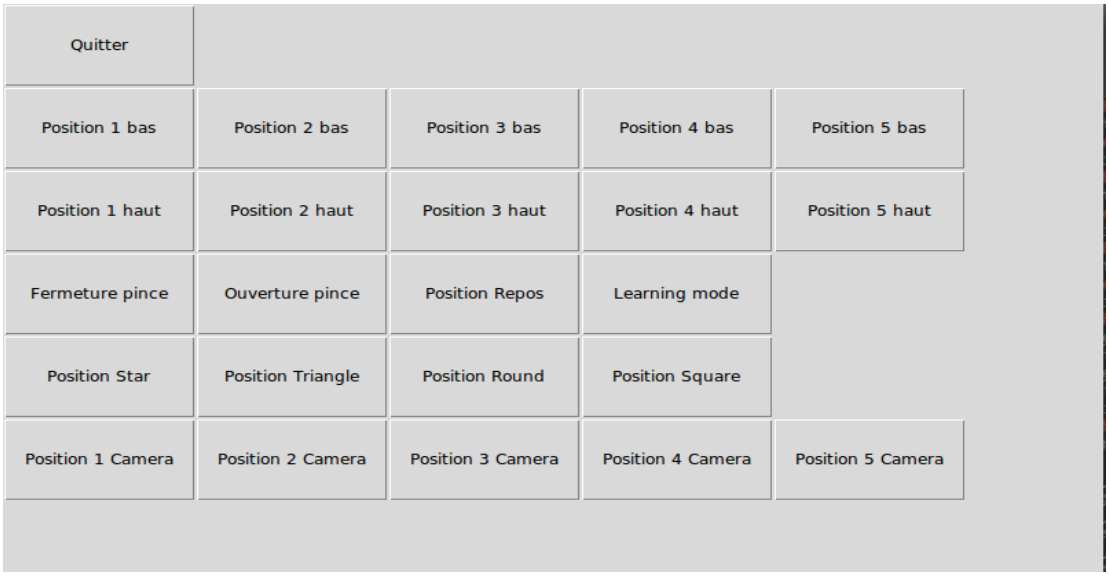


Figure 26: menu mode manuel

Cette interface est optimale pour le petit écran utilisé vu la largeur des boutons.

9.3. Raspberry :

Le Raspberry fonctionne sous le système Raspbian permettant d'intégrer la majorité des fonctionnalités disponibles sous Ubuntu (le code est testé avec succès sur une machine disposant de ce système) tout en étant très léger. Python 3.7 est installé ainsi que toutes les bibliothèques requises pour notre code (voir README.MD sur git pour tous les détails de l'installation).

9.3.1. Connexion au serveur :

En premier lieu, nous devons exécuter le code Modbus_server.py sur le Raspberry du robot afin de pouvoir envoyer des commandes de position. Ce dernier ne disposant d'aucun clavier ni écran, nous devons y accéder par SSH puis lancer le code python.

Pour simplifier l'exécution, un script connexion.sh est créé (Lien Git) celui-ci intègre la connexion SSH et la connexion par mot de passe automatique grâce à la fonction SSHPASS. Une simple exécution à ce point ne fonctionnerait pas ; en effet, celui-ci serait exécuté sur notre Raspberry plutôt que celui du robot. Pour résoudre ce problème, un changement du path est effectué par paramètre ce qui lance avec succès le serveur. La console vide s'ouvrant ne doit absolument pas être fermée sous peine de fermer le serveur. En fermant la console et donc le serveur, le client (notre Raspberry) ne peut plus communiquer.

9.3.2. Lancement programme :

En second lieu, nous lançons le code python contenant toute la gestion de la reconnaissance, du robot et de l'interface graphique par un second script programme.sh. Celui-ci, beaucoup plus simple, se charge de lancer le code python sous python3.

Le code de l'acquisition d'image est modifié par rapport au programme sur Ubuntu pour correctement utiliser le Raspberry Pi.

10. Conclusion :

Le projet a été mené à son terme malgré les soucis rencontrés avec le multithreading et le robot (communication peu évidente au début). La reconnaissance des pièces se fait à la perfection et sans aucune erreur ce qui permet de trier efficacement les pièces. Cependant le code n'a été développé que pour une seule couleur comme ce qui était prévu dans le cahier des charges.

Pour améliorer le projet et permettre de reconnaitre toutes les couleurs, la technique numéro 2 aurait dû être utilisée, car elle permet de détecter les formes qu'importe la couleur de celles-ci.

Avec plus de temps, nous aurions également pu créer un moyen pour que le robot cherche par lui-même les pièces et donc ne plus devoir placer les pièces à des endroits précis. Pour cela, il aurait fallu récupérer les positions via la caméra.

11. Source :

Installation libraire Python

Compréhension de SIFT

Forum Python

Utilisation de OPENCV

Forum Python

Communication

Script connexion

Modbus

GitHub