

Départements Génie Electrique et Informatique Industrielle

Détection et localisation de Aruco pour la robotique

RAPPORT DE STAGE Rédigé par Vinzen MALIWAT



Lu et approuvé par

- Autorise la diffusion en interne**
- N'autorise pas la diffusion, même en interne**

REMERCIEMENTS

Je souhaite exprimer ma reconnaissance envers Monsieur Christophe VERMAELEN pour son soutien et ses conseils tout au long de cette période de stage.

Je tiens également à exprimer ma gratitude envers les autres enseignants de l'IUT pour leur aide précieuse dans le bon déroulement du stage.

Monsieur Richard BRIVES est également reconnaissant d'être mon professeur référent.

PRÉSENTATION DE L'ENTREPRISE

I. Présentation générale

L’Institut Universitaire de Technologie (IUT) Nice Côte d’Azur, par la diversité de ses formations, alliant théorie et pratique, prépare efficacement depuis 50 ans ses étudiants à leur future carrière professionnelle. L’IUT Nice (rebaptisé IUT Nice Côte d’Azur depuis 1996) est une partie de l’Université Côte d’Azur (UCA). Sur le site de Fabron, il y a quatre « département » ouvert depuis 1970 :

- Génie Électrique et Informatique Industrielle
- Gestion des Entreprises et des Administrations
- Informatique
- Techniques de Commercialisation

Grâce à la collaboration entre les acteurs socio-économiques locaux et leur Institut de Technologie, d’autres départements qui ont été ouverts à partir de 1988 :

À Sophia Antipolis, l’émergence des départements

- Qualité, Logistique Industrielle et Organisation
- Réseaux et Télécommunications
- Information Communication

À Cannes, l’apparition des départements Techniques de Commercialisation, orientation Marketing du Tourisme.

Au cours des années 2000, l’offre de formation de l’IUT a été augmentée de 2 derniers départements :

- Statistique et Informatique Décisionnelle à Sophia Antipolis
- Carrières Sociales à Menton.



Figure 1: IUT Nice Côte d'Azur, vue depuis le site

En partenariat avec l'industrie et les milieux professionnels, l'IUT de Nice propose des formations menant au Bachelor Universitaire de Technologie (BUT) et à la Licence Professionnelle (LP). Le BUT, conférant le grade de licence, est aligné sur les standards internationaux, facilitant ainsi les échanges avec les universités étrangères.

Son objectif est de favoriser une insertion professionnelle rapide tout en offrant la possibilité de poursuivre des études, courtes ou longues, en école ou à l'université. L'enseignement est assuré par des enseignants-chercheurs, des enseignants et des professionnels issus des entreprises locales.

II. Mon service

Le département Génie Electrique et Informatique Industrielle (GEII) est l'un des dix départements de l'Institut Universitaire de Technologie Nice Côte d'Azur. Implémenté sur le site de Fabron, il participe à la dynamique locale depuis 1970.

Les diplômés en GEII développent divers compétences dans divers domaines tels que l'énergie, l'automatisme, l'informatique et l'électronique. Ce diplôme permet d'avoir un large choix dans une poursuite d'études pour un master ou une entrée en école d'ingénieur. Ce diplôme permet également d'avoir un choix de métiers très grands car ils touchent beaucoup de domaines.

En première année, les étudiants ont un cursus traditionnelles avec des cours magistraux, des travaux pratiques, des travaux dirigés et des projets.

En seconde années, des étudiants ont la possibilités d'aller dans une formation d'alternance qui un parcours où les étudiants sont 2 jours par semaine en entreprise et 3 jours à l'université.

Pour acquérir de l'expérience dans le monde professionnel.

En troisième années, les étudiants font tous une formation d'alternance avec une spécialisation dans trois domaines différents :

- **Automatisme et informatique industrielle (AII)**
- **Électronique et systèmes embarqués (ESE)**
- **Électricité et maîtrise de l'énergie (EME)**

L'IUT ont plusieurs contacts avec des entreprises pour les étudiants et ont des contacts dans d'autres universités. Chaque année, certains élèves ont la possibilité de faire leur étude à l'étranger. Il y a également des étudiants étrangers qui viennent faire leur étude en GEII.

Le département Génie Electrique et Informatique Industrielle propose également aux étudiants un Diplôme universitaire de technologie (DUT) pour ce qui veulent s'arrêter après leur deuxième année ou s'ils vont en écoles d'ingénieur.

Présentation du projet

I. Présentation du concours

1) Présentation générale

Le concours de robotique (ou Eurobot) est un concours de robot qui se passe chaque année et qui réunit des passionnés de robot. Chaque année le thème change et pour l'année 2025, le thème du concours est un concert.

EurobotOpen et EurobotOpen Junior sont deux événements internationaux de robotique amateur destinés aux jeunes passionnés, qu'ils soient membres d'un club, d'un groupe d'amis ou engagés dans un cadre scolaire. Leur objectif commun est de favoriser un apprentissage actif, en permettant aux participants de développer leurs savoirs, leurs compétences techniques et leur esprit d'équipe à travers un défi passionnant : la conception et la construction d'un ou plusieurs robots.

Au-delà de la compétition, ces rencontres se veulent avant tout des expériences enrichissantes, éducatives et conviviales. Elles mettent l'accent sur l'inclusivité, une valeur chère à Planète Sciences et à ses partenaires européens. Aucun modèle d'équipe n'est imposé : que l'on participe pour découvrir la robotique, pour approfondir ses connaissances ou pour relever un défi compétitif, chacun y a sa place. Les organisateurs affirment avec conviction que tous les jeunes sont les bienvenus, quelle que soit leur motivation.

Ces événements sont portés chaque année par des bénévoles de diverses nationalités, qui préparent avec enthousiasme les rencontres. Nombre d'entre eux sont d'ailleurs d'anciens participants, animés par la volonté de transmettre cette expérience unique à la nouvelle génération.

Les règlements d'EurobotOpen et d'EurobotOpen Junior ont été pensés pour être similaires, afin de faciliter l'organisation et d'offrir une continuité pédagogique entre les deux formats. Tandis qu'EurobotOpen est dédié aux robots autonomes, EurobotOpen Junior s'adresse aux moins de 18 ans et met l'accent sur des robots pilotés. Cette complémentarité permet aux organisateurs d'adapter l'événement à tous les publics, tout en conservant une cohérence technique et éducative.

Ainsi, EurobotOpen et EurobotOpen Junior s'imposent comme des expériences formatrices, accessibles et stimulantes, où les jeunes peuvent s'initier ou se perfectionner dans le domaine de la robotique, tout en partageant des moments forts, porteurs de valeurs humaines et collectives.



Figure 3: Logo de la coupe de France de robotique



Figure 2: Logo de l'Eurobot en 2025

2) Présentation du thème

Cette année, les robots se lancent dans une aventure inédite et festive : organiser davantage de matchs et de concours pour encore plus de fun ! Mais pour concrétiser ce projet ambitieux, ils doivent d'abord réunir les fonds nécessaires. C'est ainsi qu'est né le **Robot-Rock-Tour**, une série de grands concerts de charité organisés par... les robots eux-mêmes !

Mais comme tout bon spectacle, la préparation demande rigueur, coordination et esprit d'équipe. Les robots se mobilisent donc avec énergie pour relever ce nouveau défi, tout en respectant un planning serré. L'objectif est clair : que le spectacle ait lieu dans les temps, pour le plus grand plaisir de tous. Comme le dit si bien l'adage : **The show must go on !**

Les missions que devront accomplir les robots au cours de cette édition reflètent les différentes étapes de l'organisation d'un événement artistique. Elles sont à la fois techniques, stratégiques et créatives :

- **Préparer la salle de concert** : mettre en place le décor et les équipements nécessaires
- **Assurer la promotion du spectacle** : diffuser l'information pour attirer le public
- **It's show time !** : garantir que le concert se déroule sans accroc
- **Ranger les outils** : remettre tout en ordre après le spectacle
- **Estimer les entrées** : analyser le succès de l'événement en fonction du nombre de spectateurs.

À travers ces missions, les participants doivent faire preuve d'ingéniosité, d'adaptabilité et de précision. Le Robot-Rock-Tour devient ainsi un prétexte original pour explorer des problématiques robotiques concrètes dans une ambiance joyeuse et stimulante.

3) L'aire de jeu et les équipes



Figure 4: Vue générale de l'aire de jeu

L'aire de terrain est rectangle horizontale de 3m par 2m. Des boites de conserve avec une planche par-dessus sont répartis un peu partout dans le terrain. Sur le terrain, quatre Aruco markers sont présenter pour une éventuelle utilisation pour les équipes, leurs numéros sont 20, 21, 22 et 23. Sur les boites de conserve, des Aruco sont aussi placés dessus qui est le numéro 47.

Chaque équipe dispose de leur partie qui se trouve un peu partout sur la carte, il y a aussi un espace qui est hors du terrain pour un éventuelle mat pour une caméra avec la détection des Aruco markers.

Chaque robot devrait avoir un Aruco marker par-dessus qui peut avoir comme valeur entre 1 à 5 ou 6 à 10 selon l'équipe attitré.

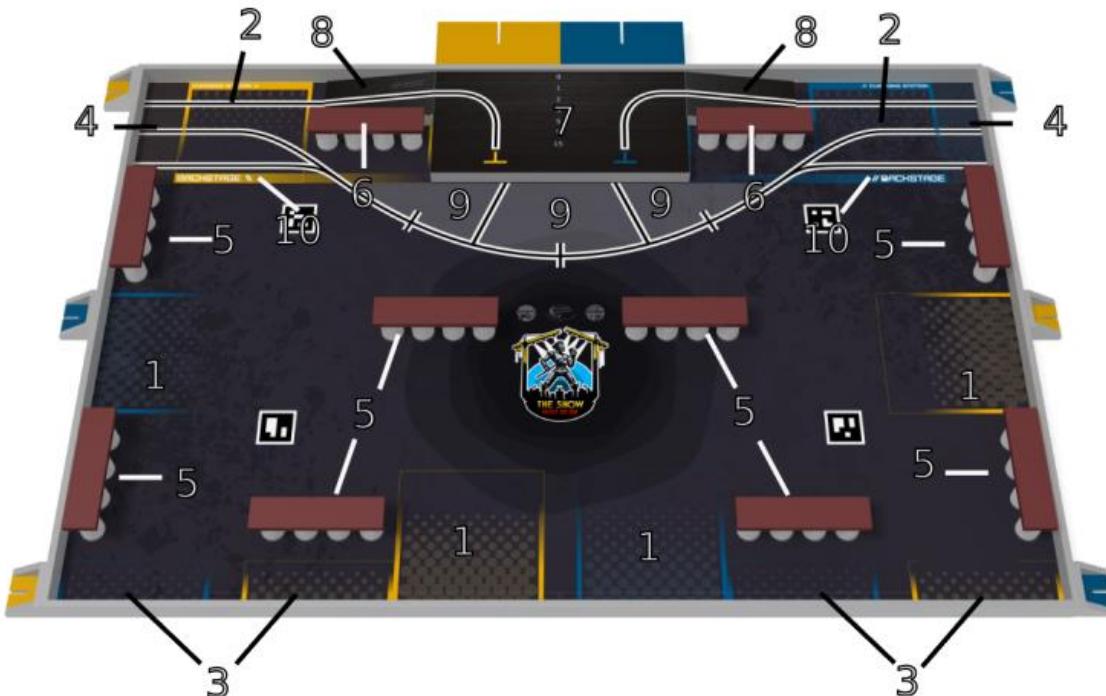


Figure 5: Vue détaillée de l'aire de jeu

- 1- Aires de départ et construction
- 2- Aires de départ et d'arrivée
- 3- Aires de construction
- 4- Aires de départ PAMI(loges)
- 5- Stock de matière première
- 6- Stock de matière première réservé
- 7- Scène
- 8- Rampe
- 9- Fosse
- 10-Zone d'arrière-scène

4) Règlement

Pour gagner il faut gagner un maximum de point.

Les équipes bleues et les équipes jaunes s'affrontent. Pour participer il faut au moins un robot fonctionnel et faire des points.

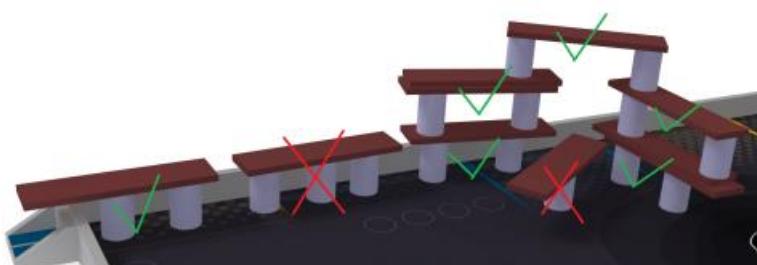


Figure 6: Example de gradin

La construction des gradins est un des moyens pour faire des points, c'est un concert donc il faut des gradins pour le public. Les gradins doivent se faire dans les zones attribuées de leurs équipes. Plus le gradin est haut et plus on gagne de point.

4 points par gradin de niveau 1.

8 points par gradin de niveau 2.

16 points par gradin de niveau 3.

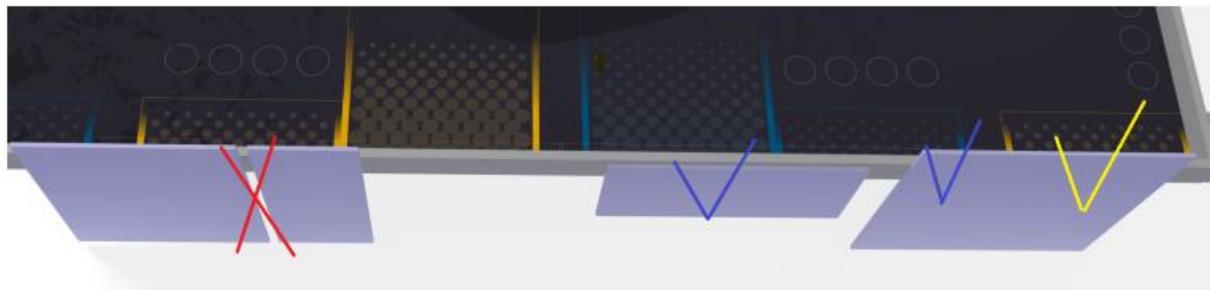


Figure 7: Exemples de banderole déployée, invalide et valide

Accrocher une banderole permet de faire beaucoup de point. La promotion d'un concert est primordiale, sans cela pas de public donc pas de concert. Pour avoir les points des banderoles, il faut déployer la banderole devant leur zone de construction. Si la banderole est valide, 20 points sont donnés.



Figure 8: Scène détaillée

Le robot principal n'est pas le seul qui peut faire des points, d'autres robots peuvent s'en occuper. Le robot qui va sur scène s'appelle « la superstar ». Si la superstar monte la rampe et va sur scène, l'équipe gagne des points, plus la superstar est proche du bord, plus on gagne de point. Et il faut des robots dans la fosse, s'ils sont tous là, cela fait aussi des points. Il faut ranger le robot principal pour que cela soit plus esthétique.

- 5 points par zone de la fosse occupée par au moins une partie de l'équipe à la fin du match.
- 5 points si la superstar de l'équipe est valide sur scène à la fin du match.
- 10 points si tous les PAMI font la fête.
- X points pour les zones de scène libre derrière la superstar en fin de match, le nombre de point dépend du numéro indiqué sur la plus haute zone libre, entre 0 et 15.
- 10 points si le robot principal de l'équipe est dans l'aire d'arrivée.

Si une équipe touche le robot de l'autre équipe, l'équipe qui touche le robot est automatiquement disqualifiée.

5) Plan de l'équipe

Dans notre équipe, nous aurons à disposition le robot principal, les PAMI et la superstar. Le robot principal ne sait que rouler et pousser des choses. Il n'est pas disposé de pince pour déplacer les boîtes de conserve et les planches. Donc pour faire les gradins nous pouvons que pousser.

Lors de la compétition, le robot principal va pousser les gradins qui sont en face de la zone de construction, va dérouler la banderole puis va directement à la zone d'arrivée. Les PAMI se déploie dans la fosse et la superstar va monter la rampe et se rapproche le plus du bord.

Si nous arrivons à faire toute ces actions, l'équipe fera 69 points.

Déroulement du stage

Lors de ma période stage, nous étions aux nombres de trois stagiaires qui était sur le projet. Le projet était supervisé par M. Vermaelen et le chef de l'équipe est un troisième année qui est Alan Panfili. L'équipe est essentiellement composés de première année et les stagiaires était juste là pour aider. Pour la préparation de ce concours, nous étions tous ensemble dans la salle 109 et travaillons chacun sur la partie qu'on lui a insigne mais l'entraide était primordiale car entre les stagiaire nous dépendions des uns des autres.



Figure 9: Mat de la caméra pour la détection

Pour ma part, je vais vous parler de la détection des Aruco markers puis de la communication qui est une partie du projet de l'un de mes camarades mais donc j'en avais besoin.

I. Raspberry Pi 4

La fondation Raspberry Pi a développé le Raspberry Pi 4, un nano-ordinateur monocarte. Dévoilé en juin 2019, il marque une évolution majeure par rapport aux générations précédentes, en ce qui concerne sa puissance, sa connectivité et sa

pratique. Elle vise avant tout à rendre l'informatique accessible au plus grand nombre, à un prix abordable.



Figure 10: Un Raspberry pi 4

Le Raspberry Pi 4 est doté d'un processeur Broadcom BCM2711, un Cortex-A72 à quatre cœurs (ARM v8) fonctionnant à 1,5 GHz. Il se décline en plusieurs variantes selon la capacité de mémoire vive : 2 Go, 4 Go ou 8 Go de RAM LPDDR4, permettant de sélectionner un modèle en fonction des exigences de performance. Par rapport aux caractéristiques clés, on trouve : 2 ports USB 3. 0 et 2 ports USB 2. 0, 1 port Ethernet Gigabit pour une connexion réseau rapide, 2 ports micro-HDMI prenant en charge la 4K, permettant une sortie sur deux écrans, 1 port GPIO de 40 broches utilisé pour relier des capteurs, des modules ou des extensions, le Wi-Fi 802. 11ac et le Bluetooth 5. 0 intégrés, ainsi qu'un lecteur de carte microSD pour le système d'exploitation et le stockage.

II. Détection des Aruco

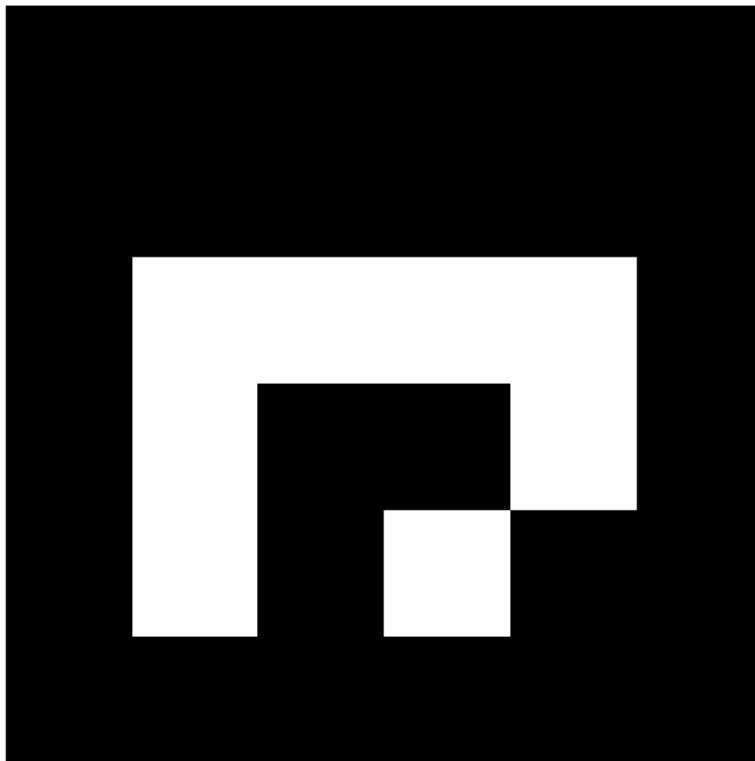


Figure 11: Aruco marker numéro 1

Lors de ce stage, j'étais chargé à la communication des Aruco markers. Mon travail permet au robot de limiter ces erreurs dans ces déplacements ainsi de mieux contrôler ces déplacements. Elle nous permettra aussi d'esquiver les robots adversaire s'ils nous gênent dans notre chemin. Pour la détection des Aruco, j'avais à disposition :

- Un Raspberry Pi 4
- Raspberry Pi Touch Display (écran)
- Pérophériques (Clavier, Souris)
- Mon ordinateur personnel
- Une carte micro-SD

Au tout début, on m'a donné le matériel pour pouvoir détecter les Aruco markers. J'ai pris connaissance du matériel et effectué des recherches. Pour pouvoir utiliser le Raspberry, il est primordial d'installer sur la carte micro-SD l'OS Raspberry Pi Imager.



Figure 12:Interface de Raspberry Pi Imager

Puis après l'installation de l'OS, l'insérer dans le Raspberry pi 4.



Figure 13:Raspberry avec une carte micro-SD

Après cela je devais configurer l'OS, en mettant la langue, les identifiants, la connexion internet et le navigateur par défaut. Il fallait par la suite activer le SPI qui sera utilisé plus

tard pour la communication et j'ai aussi activé le VNC pour pouvoir utiliser raspberry à partir de mon ordinateur personnel.

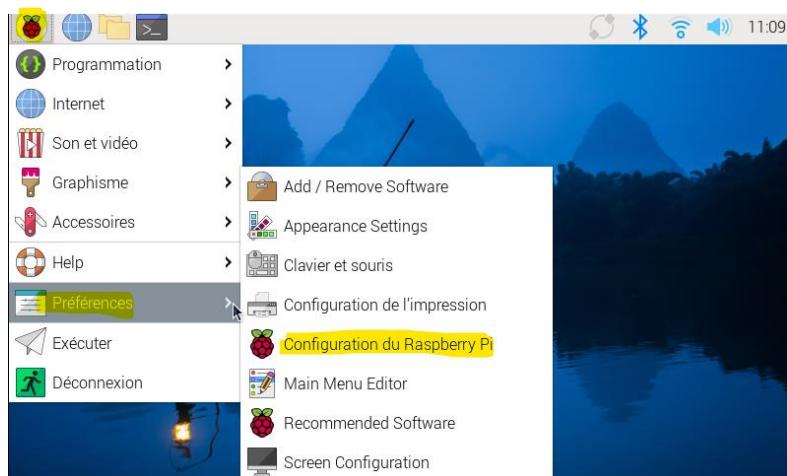


Figure 14: Interface Raspberry

Pour activer le SPI et le VNC, il fallait aller sur l'icône Raspberry en haut à droite puis aller dans Préférences < Configuration du Raspberry Pi.
Puis par la suite aller dans l'onglet interfaces, et activer le SPI et le VNC.

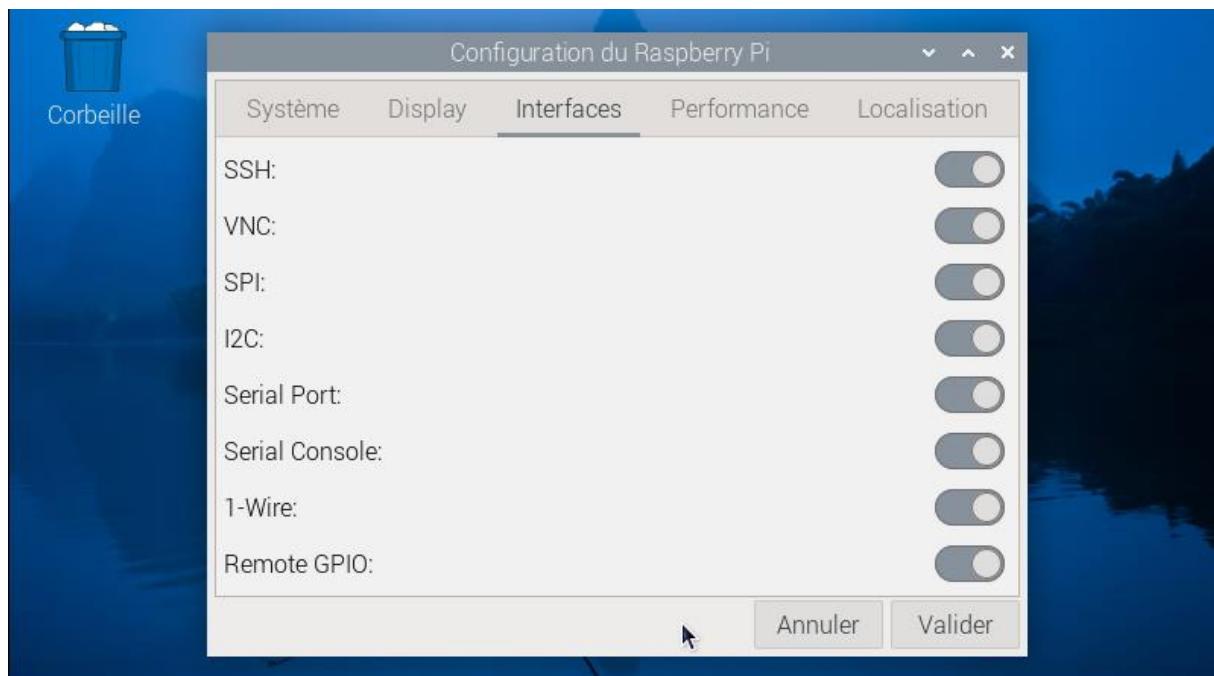


Figure 15: Activation du SPI et VNC

Ensuite, pour utiliser le Raspberry depuis mon ordinateur, j'ai utilisé le logiciel Real VNC Viewer et il fallait que mon ordinateur et le Raspberry soit connecté sur le même réseau car il fallait utiliser l'adresse IP du Raspberry. Pour connaître l'adresse IP du Raspberry, il fallait mettre le curseur de la souris sur le logo WIFI.

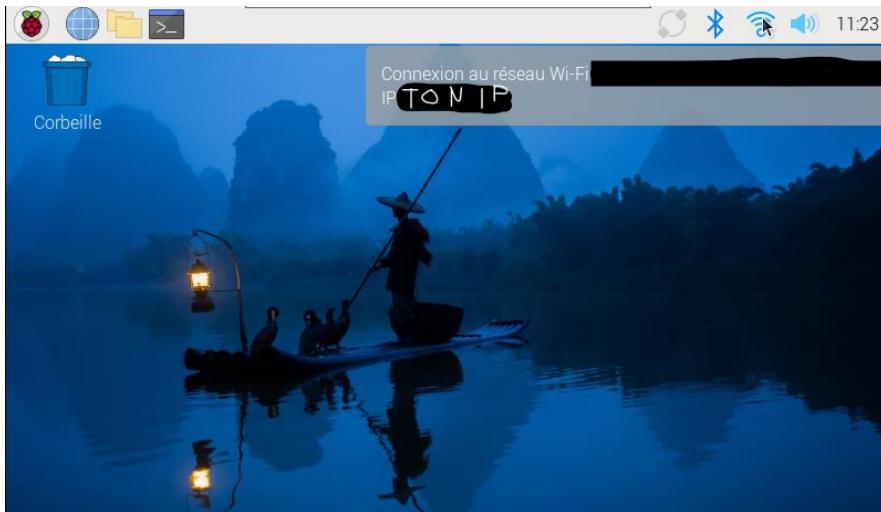


Figure 16: Adresse IP du Raspberry

Après cela, je devais aller sur le logiciel RealVNC Viewer installer sur mon ordinateur et mettre dans la barre d'adresse, l'adresse IP du Raspberry. Puis mettre les identifiants du Raspberry pour m'autoriser la connexion entre les deux appareils.

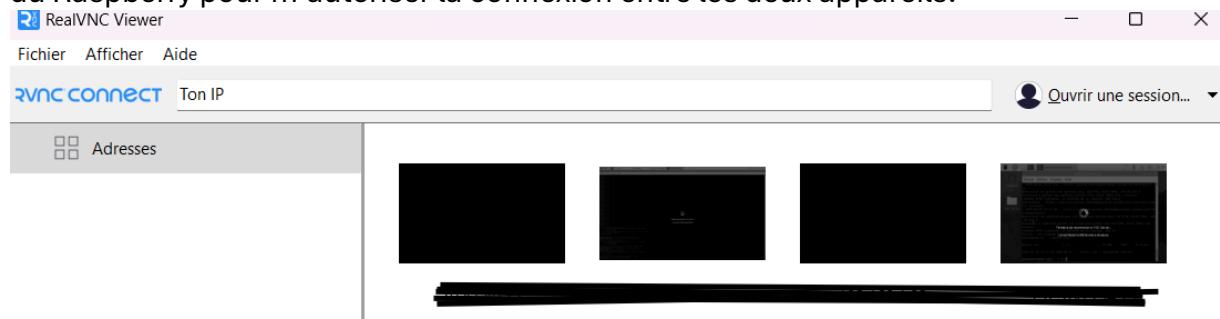


Figure 17: Interface de RealVNC Viewer

Par la suite, avant de pouvoir détecter les Aruco markers. Je devais voir si la caméra fonctionnait. J'ai pluggé la caméra dans le Raspberry puis je suis allé dans le terminal et taper la commande suivante :

```
$ rpicam-hello
```

Cette commande nous permet de savoir si la caméra fonctionne. Si la caméra fonctionnait, il y avait une transmission de ce que voyait la caméra pendant quelque seconde sinon si elle ne fonctionnait pas, il y avait un problème sur le branchement ou la caméra elle-même.

« Image à mettre du plug de la caméra avec le Raspberry »

Puisque le Raspberry fonctionné et que la caméra aussi, j'ai commencé à effectuer des recherches sur comment pouvoir lire les Aruco markers. Donc pour pouvoir les lire, il fallait utiliser la bibliothèque OpenCV qui a la fonctionnalité de détecter les Aruco markers. Au départ j'avais des problèmes d'installation car cette bibliothèque avait besoin d'autres bibliothèques pour fonctionner. Ces bibliothèques n'existaient plus car n'était plus utilisé pour le OpenCV mais je ne savais pas sur le moment. Donc pour installer OpenCV il fallait taper les commandes suivantes :

```
$ sudo apt update && sudo apt upgrade
```

Cette commande permet de mettre à jour les bibliothèques préinstallées dans l'OS et d'en rajouter pour avoir plusieurs bibliothèques à disposition.

```
$ sudo apt install cmake libgtk2.0-dev
```

Cette commande est une bibliothèque utilisée par OpenCV.

```
$ git clone https://github.com/opencv/opencv.git
```

Cette commande télécharge le code source du projet OpenCV qui est hébergé sur GitHub.

```
$ mkdir opencv/build && cd opencv/build && mkdir modules
```

La commande « mkdir » permet de créer un dossier « opencv » et un dossier « build » et « cd » va accéder dans ces dossiers.

```
$ git clone https://github.com/opencv/opencv\_contrib.git
```

En plus du dépôt principal d'OpenCV, il est nécessaire de cloner le dépôt OpenCV_contrib, qui contient des modules supplémentaires qui ne sont pas inclus par défaut dans la version de base.

```
$ mv opencv_contrib/modules/aruco modules
```

Cette commande permet de déplacer uniquement le module aruco, présent dans le dépôt OpenCV_contrib, vers le dossier local « module ». C'est ce module qui permettra à la détection des Aruco markers.

```
$ sudo make install
```

Cette commande me permet d'installer la bibliothèque OpenCV sur le système.

Après toutes ces installations, j'ai pu commencer à coder les codes qui nous permettront de détecter les Aruco markers. Pour avoir un espace propre, j'ai créé un dossier « Aruco » et fait les codes à l'intérieur.

```
$ mkdir Aruco && cd Aruco
```

Création et accès du dossier « Aruco »

Par la suite, il fallait calibrer la caméra pour pouvoir détecter les Aruco sous différent angle pour ne pas avoir de problème. Sans cela il peut y avoir une position incorrecte des Aruco et une mauvaise orientation des Aruco car la caméra peut déformer l'image qu'il peut voir dû à l'objectif.

Pour la calibration de la caméra, il fallait prendre des photos d'un damier sous différent angle.

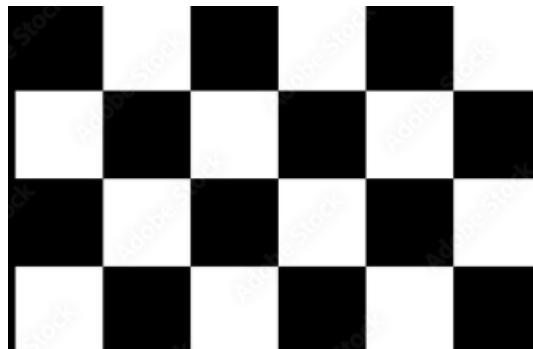


Figure 18: Un Damier

Pour cela dans le terminal, j'ai fait la commande « \$ sudo nano photo_calibrage.py » qui permet de créer le code python où je ferais des captures d'image. Voici le code de la capture d'image pour la calibration :

```
#importation des bibliothèque
from picamera2 import Picamera2
import cv2, time

#Démarre la cam
picam2 = Picamera2()
picam2.start()

prev_frame_time = time.time()

cal_image_time = 0
frame_count = 0

while True:
    frame = picam2.capture_array()
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    frame_count +=1

    if frame_count == 30:
        cv2.imwrite("cal_image_"+str(cal_image_count)+".jpg",frame)
        cal_image_count += 1
        frame_count = 0
    new_frame_time = time.time()
    fps = 1/(new_frame_time - prev_frame_time)
    prev_frame_time = new_frame_time
    cv2.putText(frame, "FPS " + str(int(fps)), (10,40), cv2.FONT_HERSHEY_PLAIN, 3, (100,255,0), 2, cv2.LINE_AA)
    cv2.imshow("Camera", frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()
```

Figure 19: Code capture d'image par caméra

Dans un premier temps, j'ai importé les bibliothèques utiles. Il y a Picamera2 qui permettra à l'activation de la caméra et à la capture d'image via la caméra. Ensuite il y a l'importation de OpenCV et de la bibliothèque time.

```
#importation des bibliothèque
from picamera2 import Picamera2
import cv2, time
```

Ensuite, j'ai initialisé la caméra en utilisant la bibliothèque Picamera2. J'ai créer un objet picam2 puis j'ai démarrer la capture de vidéo.

```
#Démarrer la cam
picam2 = Picamera2()
picam2.start()
```

L'initialisation de variable qui sera utilisé plus tard pour une boucle et prev_frame_time renvoie un temps en seconde depuis le 1^{er} janvier 1970.

```
prev_frame_time = time.time()

cal_image_count = 0
frame_count = 0
```

Puis nous avons une boucle infinie « while True » pour pouvoir faire autant de photo que l'on souhaite. L'idéale serait de au minimum une dizaine. La boucle s'arrête si on touche sur la touche Q.

```
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
```

Dans la boucle, nous avons tout d'abord une capture d'image sous forme de tableau et les bonnes couleurs à la caméra car sur la caméra les couleurs sont modifiés.

Notre peau devenait bleue car Picaméra2 utilise du RGB alors que OpenCV utilise du BGR.

```
frame = picam2.capture_array()
frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

Il y a le calcul de FPS (ou Frame Per Second) pour connaître le nombre d'image par seconde pour connaître la fluidité de la caméra. Pour calculer le nombre de FPS, on prend la nouvelle valeur capturée et on le soustrait par la précédente valeur capturée. On obtient alors la différence de temps mis entre les deux captures et en l'inversant nous obtenons le nombre d'image par seconde. La valeur obtenue est ensuite affiché sur l'écran qui est dans une fenêtre « Camera ».

```
new_frame_time = time.time()
fps = 1/(new_frame_time - prev_frame_time)
prev_frame_time = new_frame_time
cv2.putText(frame, "FPS " + str(int(fps)), (10,40), cv2.FONT_HERSHEY_PLAIN, 3, (100,255,0), 2, cv2.LINE_AA)
cv2.imshow("Camera", frame)
```

À l'intérieur de la boucle principale, un système de compte à rebours est mis en place : toutes les 30 itérations de la boucle principale, une image est automatiquement capturée. Les variables précédentes sont utilisées pour le nom des images et pour le compte à rebours.

```
if frame_count == 30:
    cv2.imwrite("cal_image_"+str(cal_image_count)+".jpg", frame)
    cal_image_count += 1
    frame_count = 0
```

Si le processus est correctement exécuté, les résultats suivants devraient être :

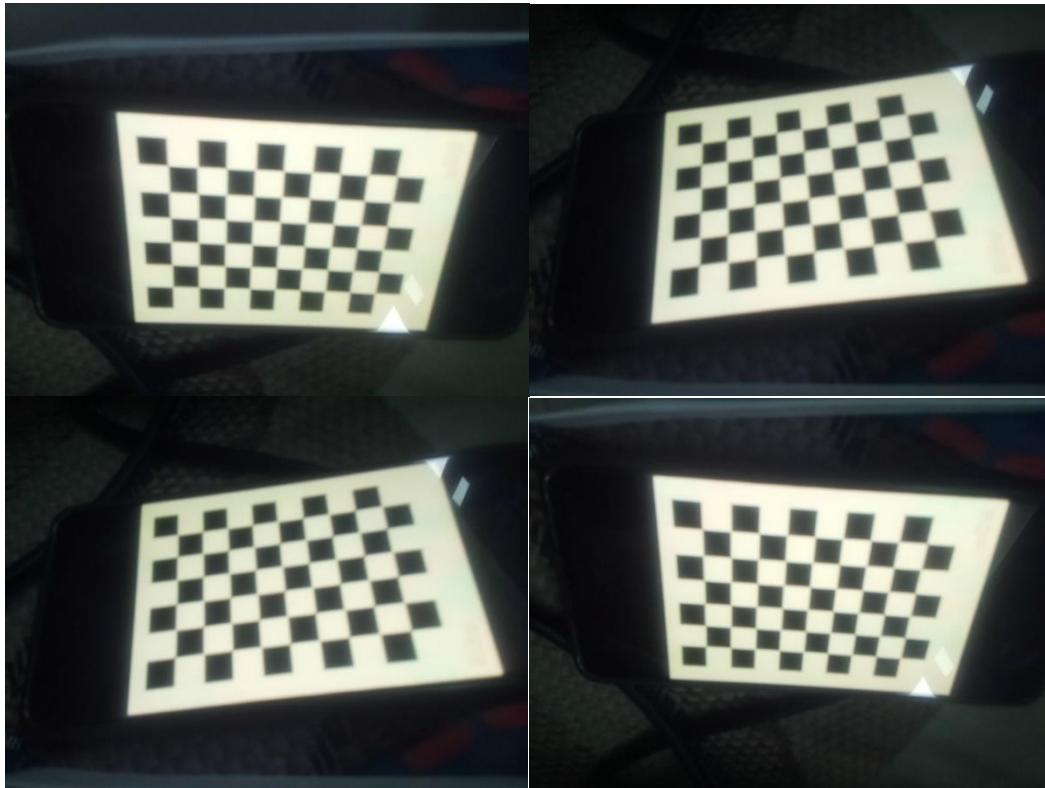


Figure 20: 4 exemples de photos

À partir des images capturées, nous allons créer un script permettant de générer un dossier de calibrage, nécessaire à la détection des marqueurs ArUco.

Le code qui permettra de faire cela est :

```
import numpy as np
import cv2
import glob

cd_width = 9
cd_height = 6
cd_square_size = 26.3

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

cd_3d_points = np.zeros((cd_width * cd_height, 3), np.float32)
cd_3d_points[:, :2] = np.mgrid[0:cd_width, 0:cd_height].T.reshape(-1, 2) * cd_square_size

list_cd_3d_points = []
list_cd_2d_img_points = []

list_images = glob.glob('*.*jpg')

for frame_name in list_images:
    img = cv2.imread(frame_name)

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    ret, corners = cv2.findChessboardCorners(gray, (9, 6), None)

    if ret == True:
        list_cd_3d_points.append(cd_3d_points)

        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
        list_cd_2d_img_points.append(corners2)

        cv2.drawChessboardCorners(img, (cd_width, cd_height), corners2, ret)
        cv2.imshow('img', img)
        cv2.waitKey(500)

cv2.destroyAllWindows()

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(list_cd_3d_points, list_cd_2d_img_points, gray.shape[::-1], None, None)

print("Calibration Matrix: ")
print(mtx)
print("Distortion: ", dist)

with open('camera_cal.npy', 'wb') as f:
    np.save(f, mtx)
    np.save(f, dist)
```

Figure 21: code de calibrage

Nous avons dans un premier temps fait l'importation des bibliothèques qui sont nécessaire. Numpy est utile pour les calculs de matrices et tableaux. Opencv pour les fonctions de calibration et traitement d'images. Glob pour lister automatiquement tous les fichiers .jpg dans le dossier.

```
import numpy as np  
import cv2  
import glob
```

Les variables utilisées sont la définition du damier. Le damier fait du 9 par 6 et la taille des carreaux devait faire 26.3mm.

```
cd_width = 9  
cd_height = 6  
cd_square_size = 26.3
```

Définit les conditions d'arrêt pour l'algorithme de détection précise des coins du damier.

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
```

Création d'une grille 3D plane représentant les coordonnées physiques des coins du damier.

```
cd_3D_points = np.zeros((cd_width * cd_height, 3), np.float32)  
cd_3D_points[:, :2] = np.mgrid[0:cd_width, 0:cd_height].T.reshape(-1, 2) * cd_square_size
```

Création des listes de points pour la calibration. Une liste pour les points 3D réels du damier et une liste de leur position dans une image en 2D.

```
list_cd_3d_points = []  
list_cd_2d_img_points =[ ]
```

Cette ligne parcourt toutes les images et va détecter le damier. Il utilise la bibliothèque Glob pour charger toutes les images .jpg qui se trouvent dans le dossier « Aruco » fait par le code précédent.

```
list_images = glob.glob('*.*.jpg')
```

```
for frame_name in list_images:  
    img = cv2.imread(frame_name)  
  
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  
  
    ret, corners = cv2.findChessboardCorners(gray, (9,6),None)
```

Si le damier est détecté dans l'image, on rajoute des points en 3D et leurs projections 2D sur l'image et on les affiche.

```
list_cd_3d_points.append(cd_3D_points)  
  
corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)  
list_cd_2d_img_points.append(corners2)  
  
cv2.drawChessboardCorners(img, (cd_width, cd_height), corners2,ret)  
cv2.imshow('img',img)  
cv2.waitKey(500)
```

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(list_cd_3d_points, list_cd_2d_img_points, gray.shape[::-1],None,None)
```

Sauvegarde les données de calibration dans un fichier appelé « camera_cal.npy » qui sera utile pour la détection des Aruco markers.

```
with open('camera_cal.npy','wb') as f:  
    np.save(f, mtx)  
    np.save(f, dist)
```

Lors de l'exécution du code, nous avons les images capturées qui défilent avec cadrillage qui se forme dans le damier. Puis quand l'exécution est terminée, un fichier camera_cal.npy apparaît.

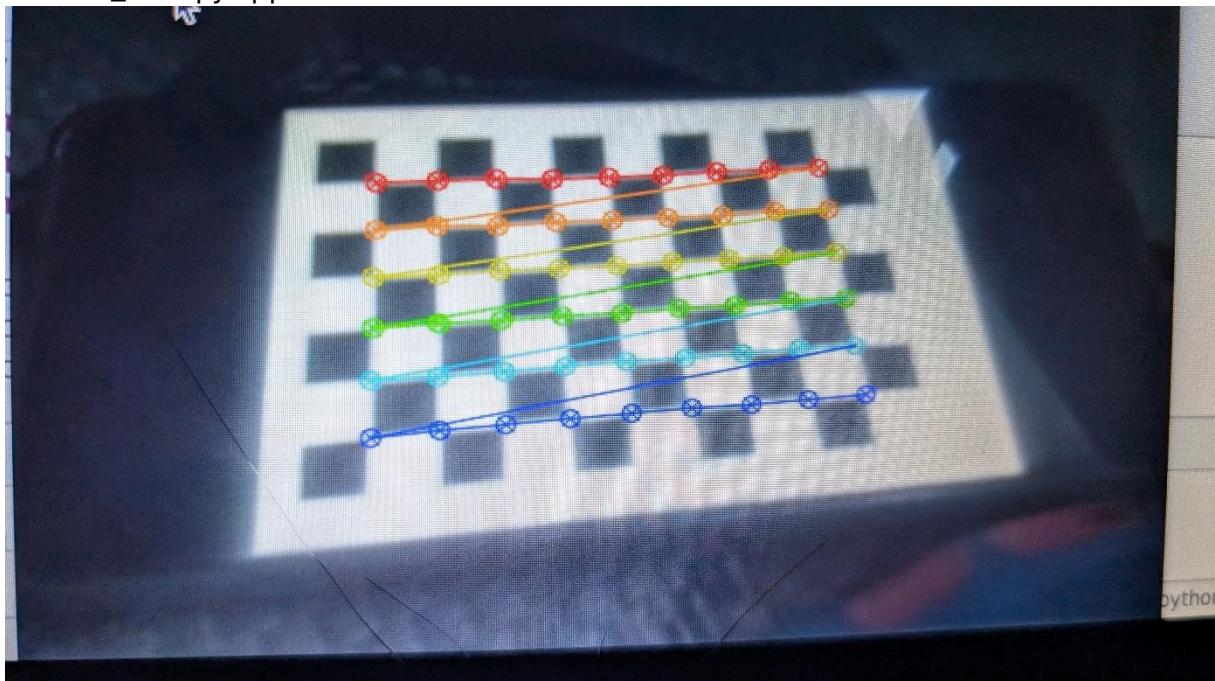


Figure 22Calibrage sur une image

Enfin nous allons passer à la détection des Aruco markers. Pour valider la détection, si un Aruco est détecté, je devais afficher son numéro sur la retransmission de la caméra

```

import numpy as np
import cv2
import cv2.aruco as aruco
from picamera2 import Picamera2
import time

marker_size = 100
with open('camera_cal.npy', 'rb') as f:
    camera_matrix = np.load(f)
    camera_distortion = np.load(f)

aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_250)

picam2 = Picamera2()
picam2.start()

while True:
    frame = picam2.capture_array()

    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, rejected = aruco.detectMarkers(gray_frame, aruco_dict, camera_matrix, camera_distortion)

    if ids is not None:
        aruco.drawDetectedMarkers(frame, corners)
        rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(corners, marker_size, camera_matrix, camera_distortion)

        for marker in range(len(ids)):
            cv2.drawFrameAxes(frame, camera_matrix, camera_distortion, rvec[marker], tvec[marker], 100)
            cv2.putText(frame, str(ids[marker][0]), (int(corners[marker][0][0][0]) - 30, int(corners[marker][0][0][1])), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 2, cv2.LINE_AA)

    cv2.imshow('frame', frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'): break
cv2.destroyAllWindows()

```

Figure 23: Code détection des Aruco marker

Ce sont toutes les bibliothèques utilisé pour la détection des Aruco. Numpy est utilisé pour le calcul des matrices et des tableaux. OpenCV pour la détection des Aruco marker et toute la bibliothèque sur les Aruco. Picaméra pour voir à travers la caméra.

Importation des bibliothèques utiles.

```

import numpy as np
import cv2
import cv2.aruco as aruco
from picamera2 import Picamera2
import time

```

Chargement de la calibration.

Le fichier .npy contient :

camera_matrix : la matrice de calibration (paramètres intrinsèques de la caméra).

camera_distortion : les coefficients de distorsion.

```

marker_size = 100
with open('camera_cal.npy', 'rb') as f:
    camera_matrix = np.load(f)
    camera_distortion = np.load(f)

```

Initialisation du dictionnaire Aruco 4x4. Il y a plusieurs Aruco de plusieurs tailles. Celle qui nous intéresse font du 4x4. Aux total, dans ce dictionnaire il y a 250 Aruco.

```
aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_250)
```

Initialisation de la caméra.

```

picam2 = Picamera2()
picam2.start()

```

Capture de la caméra pour voir en temps réel ce que voit la caméra et dans les bonnes couleurs.

```
while True:  
    frame = picam2.capture_array()  
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)  
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Détection des Aruco d'après la base de donnée du dictionnaire.

```
corners, ids, rejected = aruco.detectMarkers(gray_frame, aruco_dict, camera_matrix, camera_distortion)
```

Ce morceau de code nous indique que si un Aruco marker est détecter, sur l'image, il est contourné et on estime sa position et son orientation.

```
if ids is not None:  
    aruco.drawDetectedMarkers(frame, corners)  
    rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(corners, marker_size, camera_matrix, camera_distortion)
```

On balaye dans tout le dictionnaire les Aruco markers pour déterminer lequel est détecter par le code ensuite un repère 3D ce dessine dessus et son numéro est écrit juste à coté.

```
for marker in range(len(ids)):  
    cv2.drawFrameAxes(frame, camera_matrix, camera_distortion, rvec[marker], tvec[marker], 100)  
    cv2.putText(frame, str(ids[marker][0]), (int(corners[marker][0][0][0]) - 30, int(corners[marker][0][0][1])), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 2, cv2.LINE_AA)
```

Toute la démonstration de la détection de l'Aruco est dans une fenêtre « frame ».

```
cv2.imshow('frame', frame)
```

Pour quitter le coder, il suffit de cliquer sur q.

```
key = cv2.waitKey(1) & 0xFF  
if key == ord('q'): break
```

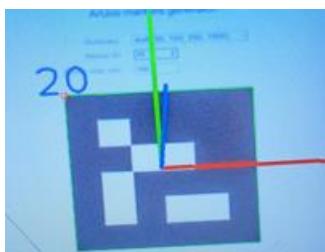


Figure 24: Démonstration d'une détection d'un Aruco

III. Communication entre Raspberry et microcontrôleur

1) SPI (Serial Peripheral Interface)

Le **SPI** est un **protocole de communication série synchrone** utilisé pour échanger des données à haute vitesse entre un microcontrôleur (comme un Raspberry Pi ou un Arduino) et un ou plusieurs périphériques (capteurs, mémoires, afficheurs, etc.).

SPI repose sur une **communication maître/esclave**. Le **maître** (souvent le microcontrôleur) contrôle la communication et envoie les signaux d'horloge. Chaque **esclave** est un composant périphérique qui répond aux ordres du maître.

Il utilise **4 lignes** principales :

- **MISO** (Master In Slave Out) : données envoyées de l'esclave vers le maître.
- **MOSI** (Master Out Slave In) : données envoyées du maître vers l'esclave.
- **SCLK** (Serial Clock) : horloge générée par le maître pour synchroniser l'échange.
- **SS/CS** (Slave Select / Chip Select) : ligne utilisée par le maître pour activer un esclave spécifique.

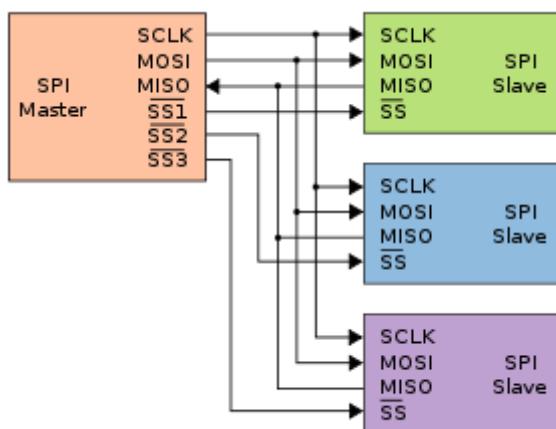
Le SPI est **pleinement duplex**, ce qui signifie que les données peuvent être échangées **dans les deux sens en même temps**. À chaque impulsion d'horloge, un bit est transféré simultanément sur MISO et MOSI.

Avantages

- **Très rapide** (bien plus que I2C dans la plupart des cas)
- **Simple à implémenter**
- **Communication stable sur de courtes distances**

Inconvénients

- Nécessite plus de fils que d'autres protocoles (notamment un fil CS par esclave)
- Pas de standardisation du format de données (configuration spécifique selon les périphériques)



2) Le module NRF24L01

Le **NRF24L01** est un module de communication sans fil utilisant la technologie **radiofréquence (RF)** dans la bande des **2,4 GHz**, conçu par Nordic Semiconductor. Il permet d'établir une communication bidirectionnelle entre deux (ou plusieurs) microcontrôleurs de manière fiable et à faible consommation.

Caractéristiques techniques

- **Fréquence** : 2,4 GHz (bande ISM, libre d'utilisation)

- **Débit** : jusqu'à 2 Mbps
- **Portée** : jusqu'à 100 mètres en extérieur (et plus avec antenne externe – version PA+LNA)
- **Alimentation** : 1,9V à 3,6V (3,3V recommandé)
- **Interface** : communication via **SPI**
- **Topologie** : point à point, étoile ou mesh simple

Fonctionnement

Le NRF24L01 fonctionne comme un **émetteur/récepteur radio** (transceiver). Il peut être configuré pour :

- envoyer des données (mode TX)
- recevoir des données (mode RX)

Les modules peuvent être appairés en définissant des **adresses logiques**. On peut ainsi créer un réseau avec plusieurs modules qui communiquent entre eux sans interférences.

Communication

Le microcontrôleur envoie les données au NRF24L01 via **SPI**. Le module se charge ensuite de la transmission radio. Côté réception, le module récupère les données et les transmet au microcontrôleur.

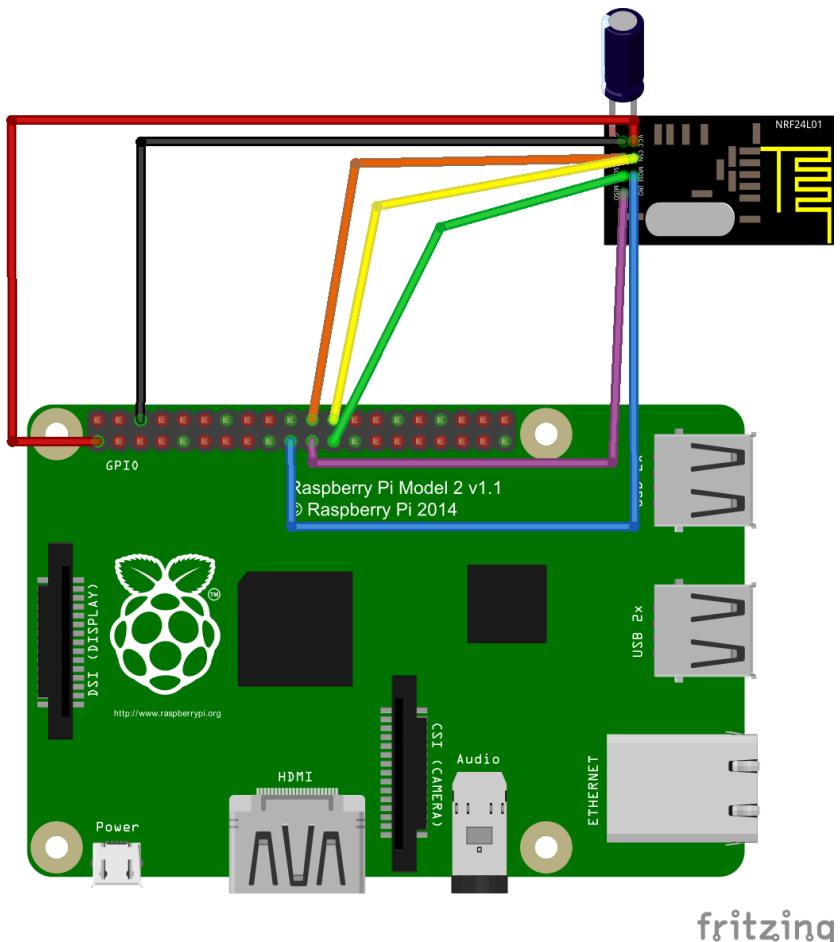
Le module supporte aussi un **accusé de réception automatique (ACK)** et la **réémission** en cas d'échec, rendant la communication plus fiable.

Avantages

- Très faible consommation
- Portée correcte en intérieur comme en extérieur
- Communication rapide, bidirectionnelle et fiable
- Facile à utiliser avec des bibliothèques comme **RF24**

Précautions

- Nécessite une **bonne alimentation stable en 3,3V**
- L'ajout d'un **condensateur de découplage (10 µF)** est souvent nécessaire pour éviter des redémarrages



fritzing

Figure 25: Câblage entre Raspberry pi 4 et nrf24l01

Pour cette partie nous étions deux stagiaires car nos objectifs était commun.
Notre objectifs était de faire communiquer un Raspberry pi 4 et un microcontrôleur.

Pour faire la communication, pour le Raspberry, il fallait installer la bibliothèque du nrf24l01 pour pouvoir utiliser ce module.

Pour ce faire, nous avons taper dans le terminal les commandes suivante :

```
$wget https://github.com/Gadgetoid/py-spidev/archive/master.zip
```

Cette commande permet de télécharger un fichier qui est dans notre cas un « master.zip ». Ce fichier est une bibliothèque utile pour la communication via NRF24L01.

```
$unzip master.zip
```

Cette ligne permet de décompresser les dossiers à l'intérieur du fichier

```
$rm master.zip
```

Cela nous permet de supprimer le fichier master.zip. Ce n'est pas obligatoire mais au cas si j'avais plus de place et pour avoir un espace de travaille plus propre.

```
$ cd py-spidev-master
```

On va dans un des fichiers qui a été dézipper.

```
$sudo python setup.py install
```

La commande sert à installer un module Python manuellement à partir de son code source, généralement contenu dans un dossier avec un fichier setup.py.

```
$cd
```

Pour retourner sur le terminal de base.

```
$cd Aruco && mkdir NRF24L01
```

Cette ligne permet de faire un dossier « NRF24L01 » dans l'espace Aruco et d'y accéder.

```
$git clone https://github.com/BLavery/lib\_nrf24
```

Permet de copier le dépôt car c'est dans ce github qu'il y a la bibliothèque pour le nrf24l01.

```
$ cd lib_nrf24 && cp lib_nrf24.py ~/Desktop/NRF24L01
```

C'est un des dossiers qu'il y avait à l'intérieur du GitHub et cp permet de copier le fichier lib_nrf24.py dans le dossier NRF24L01.

Depuis le dossier NRF24L01, le code pour la transmission des données est :

```

import numpy as np
import cv2
import cv2.aruco as aruco
from picamera2 import Picamera2
import RPi.GPIO as GPIO
import spidev
import time
from lib_nrf24 import NRF24
from math import *

spi = spidev.SpiDev()
spi.open(0,1)
spi.max_speed_hz = 2000000
GPIO.setmode(GPIO.BCM)
pipes = [[0xE7,0xE7,0xE7,0xE7,0xE7],[0xF0,0xF0,0xF0,0xF0,0xF0]]
radio = NRF24(GPIO, spi)
radio.begin(1,23)
radio.powerUp()
radio.setPayloadSize(32)
radio.setChannel(2)
radio.setDataRate(NRF24.BR_1MBPS)
radio.stopListening()
radio.ce(1)
radio.openWritingPipe(pipes[0])
radio.printDetails()
def envoyer_message(message):
    message_code = list(message.encode("utf-8"))
    while len(message_code) < 32:
        message_code.append(0)
    print(f"{message_code}")
    radio.write(message_code)

marker_size = 100
with open('camera_cal.npy', 'rb') as f:
    camera_matrix = np.load(f)
    camera_distortion = np.load(f)

aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_250)

picam2 = Picamera2()
config = picam2.create_preview_configuration({"size": (820, 616)})
picam2.configure(config)
picam2.start()
picam2.set_controls({"ScalerCrop": (0, 0, 4608, 2592)})

x20, y20= 0 , 0
x21, y21= 0 , 0
x22, y22= 0 , 0
x23, y23= 0 , 0

```

```

while True:
    time.sleep(0.5)
    frame = picam2.capture_array()
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, rejected = aruco.detectMarkers(gray_frame, aruco_dict, camera_matrix, camera_distortion)

    if ids is not None:
        aruco.drawDetectedMarkers(frame, corners)

        rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(corners, marker_size, camera_matrix, camera_distortion)

        for marker in range(len(ids)):
            marker_id = ids[marker][0]

            cv2.drawFrameAxes(frame, camera_matrix, camera_distortion, rvec[marker], tvec[marker], 100)
            cv2.putText(frame, str(ids[marker][0]), (int(corners[marker][0][0][0]) - 30, int(corners[marker][0][0][1])), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 2, cv2.LINE_AA)

            x_coord = tvec[marker][0][0]
            y_coord = tvec[marker][0][1]*(-1)

            x_coord_format = "{:.2f}".format(x_coord)
            y_coord_format = "{:.2f}".format(y_coord)
            tvec_str = "x:{}{}y:{}{}\n".format(x_coord_format, y_coord_format)

            #time.sleep(0.1)

            if marker_id == 20:
                x20, y20= x_coord_format, y_coord_format
                #print(f"x20:{x20} y20:{y20}")

            if marker_id == 21:
                x21, y21= x_coord_format, y_coord_format
                #print(f"x21:{x21} y21:{y21}")

            if marker_id == 22:
                x22, y22= x_coord_format, y_coord_format
                #print(f"x22:{x22} y22:{y22}")

            if marker_id == 23:
                x23, y23= x_coord_format, y_coord_format
                #print(f"x23:{x23} y23:{y23}")

            if marker_id == 1:
                print("Coord1{}tvec_str")
                envoyer_message(tvec_str)

            if marker_id == 2:
                print("Coord2{}tvec_str")
                envoyer_message(tvec_str)

# rotation_matrix, _ = cv2.Rodrigues(rvec[marker])
angle_2d = np.arctan2(rotation_matrix[1, 0], rotation_matrix[0, 0])
angle_2d_deg = np.degrees(angle_2d)
format_angle2d=" {:.2f} ".format(angle_2d_deg)
if marker_id == 7:
    print(tvec_str)
    envoyer_message(tvec_str)
    dist1=sqrt((int(float(x20))-int(float(x_coord_format)))*2+(int(float(y20))-int(float(y_coord_format)))*2)
    dist2=sqrt((int(float(x21))-int(float(x_coord_format)))*2+(int(float(y21))-int(float(y_coord_format)))*2)
    dist3=sqrt((int(float(x22))-int(float(x_coord_format)))*2+(int(float(y22))-int(float(y_coord_format)))*2)
    dist4=sqrt((int(float(x23))-int(float(x_coord_format)))*2+(int(float(y23))-int(float(y_coord_format)))*2)
    dist1format=" {:.2f} ".format(dist1)
    dist2format=" {:.2f} ".format(dist2)
    dist3format=" {:.2f} ".format(dist3)
    dist4format=" {:.2f} ".format(dist4)
    print("angle{}format_angle2d")
    envoyer_message(format_angle2d)

    if dist1<=dist2 and dist1<=dist3 and dist1<=dist4:
        print("dist1: {}dist1format")
        envoyer_message("dist1: "+dist1format)
        #time.sleep(0.1)

    elif dist2<=dist1 and dist2<=dist3 and dist2<=dist4:
        print("dist2: {}dist2format")
        envoyer_message("dist2: "+dist2format)
        #time.sleep(0.1)

    elif dist3<=dist2 and dist3<=dist1 and dist3<=dist4:
        print("dist3: {}dist3format")
        envoyer_message("dist3: "+dist3format)
        #time.sleep(0.1)

    else:
        print("dist4: {}dist4format")
        envoyer_message("dist4: "+dist4format)
        #time.sleep(0.1)

cv2.imshow('frame', frame)

key = cv2.waitKey(1) & 0xFF
if key == ord('q'):
    break

cv2.destroyAllWindows()

```

Figure 26 : Code de la transmission des données

Pour ce code nous avons importés, les bibliothèques qu'on avait besoin pour les Aruco markers mais aussi pour le NRF24L01 pour la communication.

```
import numpy as np
import cv2
import cv2.aruco as aruco
from picamera2 import Picamera2
import RPi.GPIO as GPIO
import spidev
import time
from lib_nrf24 import NRF24
from math import *
```

On a initialisé le module NRF24L01 en lui configurant :

- Son adresse ([0xE7,0xE7,0xE7,0xE7,0xE7])
- Les broches GPIO où il est connecté (CE = 1, CSN = 23)
- Son canal de communication (2)
- La taille des messages qui peut envoyer (32 octets)
- Son débit de transmission(1Mbps)

Il ne doit jamais recevoir de données et doit seulement en transmettre.

```
spi = spidev.SpiDev()
spi.open(0,1)
spi.max_speed_hz = 2000000
GPIO.setmode(GPIO.BCM)
pipes = [[0xE7,0xE7,0xE7,0xE7,0xE7],[0xF0,0xF0,0xF0,0xF0,0xF0]]
radio = NRF24(GPIO, spi)
radio.begin(1,23)
radio.powerUp()
radio.setPayloadSize(32)
radio.setChannel(2)
radio.setDataRate(NRF24.BR_1MBPS)
radio.stopListening()
radio.ce(1)
radio.openWritingPipe(pipes[0])
radio.printDetails()
```

J'ai fait une fonction d'envoie pour faciliter le code. Lorsque cette fonction est appelé, il envoie directement via NRF24L01 les données qu'on veut.

```
def envoyer_message(message):
    message_code = list(message.encode("utf-8"))
    while len(message_code) < 32:
        message_code.append(0)
    print(f"{message_code}")
    radio.write(message_code)
```

Initialisation de la caméra et des paramètres Aruco

```
marker_size = 100
with open('camera_cal.npy', 'rb') as f:
    camera_matrix = np.load(f)
    camera_distortion = np.load(f)

aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_4X4_250)
```

Démarrage et configuration de la caméra. En effet nous avons configurer la caméra car si nous le faisions pas, nous étions incapable de toute les détecter car la configuration par défaut zoomer trop.

```
picam2 = Picamera2()
config = picam2.create_preview_configuration({"size": (820, 616)})
picam2.configure(config)
picam2.start()
picam2.set_controls({"ScalerCrop": (0, 0, 4608, 2592)})
```

Variable par défaut utilisé plus tard dans le code.

```
x20, y20= 0 , 0  
x21, y21= 0 , 0  
x22, y22= 0 , 0  
x23, y23= 0 , 0
```

Capture d'image dans la boucle pour obtenir une vidéo dans les bonnes couleurs.

```
frame = picam2.capture_array()  
frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)  
gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Détection des Aruco marqueur avec leur numéro donné à l'image.

```
corners, ids, rejected = aruco.detectMarkers(gray_frame, aruco_dict, camera_matrix, camera_distortion)  
  
if ids is not None:  
    aruco.drawDetectedMarkers(frame, corners)  
    rvec, tvec, _objPoints = aruco.estimatePoseSingleMarkers(corners, marker_size, camera_matrix, camera_distortion)  
  
    for marker in range(len(ids)):  
        marker_id = ids[marker][0]  
  
        cv2.drawFrameAxes(frame, camera_matrix, camera_distortion, rvec[marker], tvec[marker], 100)  
        cv2.putText(frame, str(ids[marker][0]), (int(corners[marker][0][0][0]) - 30, int(corners[marker][0][0][1])), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 2, cv2.LINE_AA)
```

Ces lignes de code nous permet de connaître la position des Aruco markers qui ont tous une coordonnée.

```
x_coord = tvec[marker][0][0]  
y_coord = tvec[marker][0][1]*(-1)  
  
x_coord_format = "{:.2f}".format(x_coord)  
y_coord_format = "{:.2f}".format(y_coord)
```

Je cherche à déterminer les coordonnées des Arucos qui sont dans sur le terrain pour mieux savoir où est le robot par rapport à eux.

```
if marker_id == 20:  
    x20, y20= x_coord_format, y_coord_format  
    #print(f"x20:{x20} y20:{y20}")  
  
if marker_id == 21:  
    x21, y21= x_coord_format, y_coord_format  
    #print(f"x21:{x21} y21:{y21}")  
if marker_id == 22:  
    x22, y22= x_coord_format, y_coord_format  
    #print(f"x22:{x22} y22:{y22}")  
  
if marker_id == 23:  
    x23, y23= x_coord_format, y_coord_format  
    #print(f"x23:{x23} y23:{y23}")
```

J'ai pris l'initiative de prendre comme référence les Aruco 1 et 2 comme robot ennemie. On aura besoin de leur coordonnée pour les esquivée.

```
if marker_id == 1:  
    print(f"Coord1{tvec_str}")  
    envoyer_message(tvec_str)  
  
if marker_id == 2:  
    print(f"Coord2{tvec_str}")  
    envoyer_message(tvec_str)
```

Pour connaître l'orientation d'un Aruco marker.

```
rotation_matrix, _ = cv2.Rodrigues(rvec[marker])  
angle_2d = np.arctan2(rotation_matrix[1, 0], rotation_matrix[0, 0])  
angle_2d_deg = np.degrees(angle_2d)  
format_angle2d = "{:.2f}".format(angle_2d_deg)
```

Dans notre cas nous sommes le robot principale avec comme numéro le 7. Je dois connaitre ma position, mon orientation et je suis proche de quelle Aruco sur le terrain pour savoir à peu près où j'en suis. J'ai dû utiliser une formule pour savoir par rapport à mes coordonnées j'étais le plus proche de quelle Aruco marker.

```

if marker_id == 7:
    print(tvec_str)
    envoyer_message(tvec_str)
    dist1=sqrt((int(float(x20))-int(float(x_coord_format)))*2+(int(float(y20))-int(float(y_coord_format)))*2)
    dist2=sqrt((int(float(x21))-int(float(x_coord_format)))*2+(int(float(y21))-int(float(y_coord_format)))*2)
    dist3=sqrt((int(float(x22))-int(float(x_coord_format)))*2+(int(float(y22))-int(float(y_coord_format)))*2)
    dist4=sqrt((int(float(x23))-int(float(x_coord_format)))*2+(int(float(y23))-int(float(y_coord_format)))*2)
    dist1format=" {:.2f} ".format(dist1)
    dist2format=" {:.2f} ".format(dist2)
    dist3format=" {:.2f} ".format(dist3)
    dist4format=" {:.2f} ".format(dist4)
    print(F"angle{format_angle2d}")
    envoyer_message(format_angle2d)

    if dist1<=dist2 and dist1<=dist3 and dist1<=dist4:
        print(F"dist1: {dist1format}")
        envoyer_message("dist1: "+dist1format)
        #time.sleep(0.1)
    elif dist2<=dist1 and dist2<=dist3 and dist2<=dist4:
        print(F"dist2: {dist2format}")
        envoyer_message("dist2: "+dist2format)
        #time.sleep(0.1)

    elif dist3<=dist2 and dist3<=dist1 and dist3<=dist4:
        print(F"dist3: {dist3format}")
        envoyer_message("dist3: "+dist3format)
        #time.sleep(0.1)

    else:
        print(F"dist4: {dist4format}")
        envoyer_message("dist4: "+dist4format)
        #time.sleep(0.1)

```

Fermeture du programme si l'on touche sur la touche « q ».

```

cv2.imshow('frame', frame)

key = cv2.waitKey(1) & 0xFF
if key == ord('q'): break

cv2.destroyAllWindows()

```

Conclusion

Pour conclure, j'ai pu atteindre mon objectif qui est de détecter les Aruco markers. J'ai pu voir comment travailler en équipe car entre stagiaire nous devions avancer au même rythme et nous aider pour avancer. La communication était primordiale pour l'équipe et aussi dans le projet car sans la communication ma partie aurait servi à rien.