



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Projet de semestre 5 / 2020-2021

Filière Informatique

Eurobot - Système de localisation basé sur la vision par ordinateur

Documentation de projet

09.11.20 – Version 0.25

Denis Rosset

Superviseurs :

Jacques Supcik
Nicolas Schroeter

Hes·so

Haute Ecole Spécialisée
de Suisse occidentale

Fachhochschule Westschweiz

Table des versions

Version	Date de publication	Auteur	Description
0.1	09.11.2020	Denis Rosset	Ébauche de la documentation du premier sprint
0.1.1	09.11.2020	Denis Rosset	Mise à jour des points concernant l'analyse des caméras et des systèmes embarqués
0.2	20.01.2021	Denis Rosset	Refonte de la structure
0.21	25.01.2021	Denis Rosset	Correction de la structure et ajout de points
0.22	26.01.2021	Denis Rosset	Ecriture de la spécification et de la conception et ajout d'un paragraphe d'analyse sur l'interprétation d'une image 3d et la calibration d'une caméra
0.23	26.01.2021	Denis Rosset	Ecriture du paragraphe de la réalisation, des tests et validation, des améliorations futures, de la conclusion
0.24	28.01.2021	Denis Rosset	Ecriture des chapitres de mise en place, écriture de l'annexe et de la déclaration sur l'honneur, ajout de chapitres dans tests et validation et correction de phrases à divers endroits du rapport
0.25	28.01.2021	Denis Rosset	Correction des erreurs d'orthographe

Table des matières

1	Introduction.....	4
1.1	Acteurs.....	4
1.2	Contexte	4
1.3	Situation avant le projet	4
1.4	Objectif du projet	5
1.5	Méthode de travail	5
1.6	Structure du rapport WIP	7
2	Analyse	8
2.1	Analyse de risques.....	8
2.2	Choix de l'algorithme.....	9
2.3	Interprétation 3d d'une image et calibration d'une caméra.....	11
2.4	Choix du système embarqué	13
2.5	Choix de caméra	15
3	Spécification	17
3.1	Organisation physique.....	17
3.2	Positions spécifiques des ArUcos	17
3.3	Configuration du système	18
3.4	Paramètres du système	19
3.5	Protocole de communication	19
4	Conception	20
4.1	Fonctionnement du programme	20
4.2	Récupération de la position du robot.....	21
4.3	Conception du support.....	23
4.4	Choix du meilleur ArUco par robot.....	24
4.5	Utilisation d'un échantillon de données pour augmenter la précision.....	24
5	Réalisation	25
5.1	Environnement de développement et logiciels utilisés	25
5.2	Implémentation du programme.....	25
5.3	Gestion de la déformation de l'objectif de la caméra.....	28
5.4	Réalisation du support.....	29
5.5	Outils Réalisés et utilisés pendant le projet	29
5.6	Alimentation du Jetson Nano	30
6	Tests et validation.....	31
6.1	Test fonctionnel.....	31
6.2	Précision et latence	31

7	Améliorations et travaux futurs	36
7.1	Choix de l'algorithme de détection des marqueurs ArUcos.....	36
7.2	Choix de la caméra	36
7.3	Utilisation de plusieurs caméras	36
7.4	Amélioration des performances.....	36
7.5	Changement des marqueurs présent sur les robots	37
7.6	Autres méthodes de détection.....	37
8	Mise en place et exemple d'utilisation.....	39
8.1	Mise en place et installation du matériel	39
8.2	Utilisation du programme	40
8.3	Résultat observé sur les robots	40
9	Conclusion	42
9.1	Comparaison entre les résultats et les objectifs	42
9.2	Rétrospective sur la planification initiale	42
9.3	Conclusion personnelle	42
10	Déclaration sur l'honneur.....	44

1 Introduction

Depuis plusieurs années, la HEIA-FR participe au concours de robotique Eurobot. Le but du concours change chaque année. Il est communiqué via le règlement [1].

Ce concours réunit des étudiants des filières informatique, génie électrique et génie mécanique, dans le but de réaliser deux robots capables d'effectuer différentes tâches. Durant chaque manche, deux équipes se voient attribuer une couleur puis s'affrontent pendant une durée 100 secondes durant lesquelles les robots se déplacent sur le terrain en même temps. Aucun contact n'est permis entre les robots et aucune tentative d'entraver le bon fonctionnement des systèmes ennemis n'est autorisé. La nécessité de connaître la position des robots (alliés et adverses) est primordiale afin de prévoir les déplacements et les actions à effectuer.

1.1 Acteurs

Ce projet est suivi par les personnes suivantes :

- Jacques Supcik, *Superviseur*
- Nicolas Schroeter, *Superviseur*
- Denis Rosset, *Etudiant*

1.2 Contexte

La filière informatique a déjà développé plusieurs solutions pour connaître la position des robots sur le plateau, mais ces méthodes souffraient de limites en termes d'utilisation ou de précision. Le but de ce projet est de créer un système de localisation à partir d'un système composé d'une ou plusieurs caméras posées sur le mât central (figure 1 rond rouge) et de marqueurs présents sur les robots et sur la table (figure 1 rond violet). Ce mât central permet d'avoir une vue totale de l'aire de jeu.



Figure 1 - Vue générale de l'aire de jeu

1.3 Situation avant le projet

Actuellement, le robot se repère grâce à deux roues codeuses parallèles aux roues d'entraînement. Ces roues codeuses permettent de savoir quelle distance est effectuée par le robot et ainsi connaître sa

position sur l'aire de jeu. Ce système souffre du fait que sa calibration n'est réalisée qu'une fois au début du match et donc perd en précision de manière linéaire par rapport à la distance parcourue par le robot.

Un second système de localisation est actuellement présent sur un des deux robots. Il s'agit d'un LIDAR qui, couplé à un algorithme, permet de mesures à quelle distance des bords de la table se trouve le robot. Le système est précis au demi-centimètre et peut être calibré à tous moments mais nécessite qu'il n'y ait pas d'objets entre le mur et lui-même.

Aucune des deux solutions ne permet de détecter la présence d'ennemis sur le terrain et ces systèmes sont limitées en précision ou en praticité.

1.4 Objectif du projet

L'objectif du projet est un système composé d'une ou plusieurs caméras et d'un système embarqué communiquant sans fil. Ce système permet de connaître la position des robots tout en respectant les contraintes imposées par le règlement et par l'équipe Eurobot RTFM.

1.4.1 Contraintes imposées par le règlement

- Le système de caméras doit tenir sur le mat central et se contraindre aux dimensions imposées par le règlement (point G.4)
- La communication sans fil du système doit être en adéquation avec le règlement (point G.6)

1.4.2 Contraintes imposées par l'équipe Eurobot RTFM

- Le traitement de l'image est directement effectué sur le mat à côté de la caméra grâce au système embarqué
- L'identification de la position du robot se fait grâce aux marqueurs¹ (point G.7)
- Les informations sont envoyées à l'infrastructure de l'équipe RTFM avec une communication sans fil
- Les informations à envoyer sont la position de chaque robot en coordonnées x/y en mm par rapport à l'aire de jeu ainsi que l'orientations de chaque robot
- Le système développé doit pouvoir être monté et démonté entre chaque manche du concours
- Le système doit permettre une détection rapide avec une précision de moins de 5cm et une latence de moins de 1000ms
- Le système doit permettre une détection lente et précise des robots avec une précision de moins de 2cm et une latence de moins de 5000ms

1.5 Méthode de travail

Afin de réaliser ce projet, un suivi constant est fait par les superviseurs grâce à des séances de projet qui ont lieu chaque semaine auxquelles les superviseurs et l'étudiant ont participé. Un procès-verbal a été rédigé pour chaque séance¹.

1.5.1 Cahier des charges

Un cahier des charges a été écrit au début du projet pour clarifier les besoins et objectifs relatif au projet. Ce cahier a permis de connaître chaque buts et contraintes nécessaires à la réalisation de ce projet. Une

¹ <https://gitlab.forge.hefr.ch/denis.rosset/sauron2021/-/tree/master/doc/pvs>

comparaison entre les objectifs du cahier des charges et les objectifs atteints dans le projet est disponible dans le chapitre 9.1.

1.5.2 Planification initiale

La planification initiale du projet a été réalisée en même temps que le cahier des charges

	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18
Sprint 1 Analyse primaire																	
Cahier des charges																	
Validation cahier des charges			VE														
Défense				ME													
Analyse de risques																	
Étude de différentes caméras																	
Étude de différents systèmes embarqués																	
Etude de différents algorithmes																	
Sprint 2 prototype sans communication																	
Tests des éléments hardware lors de la compétition							SA										
Calibration extrinsèque du système par rapport à l'aruco présent au milieu d e l'air de jeu																	
Etude de types de transmissions des données																	
Sprint 3 prototype avec communication																	
Détection de la position de l'aruco présents sur un des robots																	
Définir le protocole de transmission des données																	
Implémenter le protocole de transmission des données																	
Sprint 4 produit final, tests, documentation, défense																	
Concevoir un support efficace pour installer la caméra																	
Tester la fiabilité du hardware																	
Tester la précision et la fiabilité du software																	
Documentation																JE	
Défense orale																	ME

Figure 2 Planification initiale

Une rétrospective sur la planification et le déroulement du projet a été réalisée dans le chapitre 9.2.

1.6 Structure du rapport WIP

Ce rapport est séparé en 10 chapitres. Chaque chapitre contient des sous chapitres qui expliquent plus en détails les thèmes. La liste des chapitres est décrite ci-dessous :

- 1. Introduction**
Explique le contexte du projet, ainsi que les buts à atteindre et le planning initial
- 2. Analyse**
Définit les risques, et documente les différents choix matériels et technologiques réalisés durant le projet
- 3. Spécification**
Spécifie l'organisation physique des composants et la manière dont ils interagissent entre eux
- 4. Conception**
Indique la structure de l'implémentation, l'utilisation des algorithmes ainsi que le protocole de communication et les nécessités hardware
- 5. Réalisation**
Explique les éléments importants de l'implémentation les différentes parties du projet ainsi que comment les composants du projet ont été réalisé
- 6. Tests et validation**
Contient les protocoles de tests utilisés et les résultats de ces derniers
- 7. Mise en place et exemple d'utilisation**
Indique les étapes à réaliser pour mettre en place et utiliser le projet ainsi qu'un mode d'emploi sans les explications techniques pour les utilisateurs
- 8. Améliorations et travaux futurs**
Contient une description des options qui n'ont été assez approfondies pour être décrites dans l'analyse ainsi que les idées d'amélioration qui ont été découvertes pendant le projet mais n'ont pas pu être implémentées
- 9. Conclusion**
Donne une rétrospective sur la planification et sur les buts du projet, ainsi qu'une conclusion personnelle
- 10. Déclaration sur l'honneur**
Contient la déclaration sur l'honneur signée

2 Analyse

Ce chapitre décrit comment le projet a été analysé et comment ont été étudiés les besoins matériels et logiciels.

2.1 Analyse de risques

Une analyse de risque a été effectuée tôt dans le projet afin de se rendre compte des problèmes externes qui pourront être rencontrés (Figure 3).

		Chance d'occurrence			
		1%	5%	20%	50%
Système inutilisable		P5	P4, P7	P7	
Système fonctionne partiellement ou grosse perte de précision (<15cm)		P2		P3	
Système continue de fonctionner, perte de précision modéré (<5cm)			P1	P8	
Système continue de fonctionner, légère perte de précision (<1cm)				P6	
ID	Risque				
P1	Intensité lumineuse trop faible				
P2	Système ébloui par l'adversaire				
P3	Déplacement involontaire du système (vibrations)				
P4	Problème de communication avec le robot				
P5	Problème de fonctionnement du hardware				
P6	Mauvaise installation du système (légèrement décalé)				
P7	Mauvaise installation du système (extrêmement décalé)				
P8	Arucos mal imprimés / mal collés				

Figure 3 Matrice de risques

Grâce à cette analyse de risques, nous pouvons nous rendre compte des problèmes auxquels va être confronté le projet et ainsi les prévoir et les éviter.

2.2 Choix de l'algorithme

Le but principal du projet est de détecter la position d'ArUcos sur le terrain grâce à une caméra. L'algorithme qui est testé et utilisé dans les benchmarks est décrit dans ce paragraphe. Il est basé sur les techniques décrites dans [2].

2.2.1 Résumé des publications scientifiques

OpenCV fournit une implémentation complète pour la détection des marqueurs ArUco [2] et pour l'impression de ceux-ci [3].

Voici un résumé de l'article de détection des marqueurs [2] :

Plusieurs filtres sont appliqués sur l'image afin de pouvoir savoir quels endroits de l'image sont des marqueurs et lesquels sont des informations qui ne nous intéressent pas.

(Figure 2) Un algorithme de seuillage est appliqué à l'image originale (a) pour arriver à l'image (b) puis une extraction des contours est faite (c). Les polygones à quatre côtés détectés sont filtrés afin de se concentrer sur les zones intéressantes (d). Une image canonique est calculée pour un des carrés (e) puis la méthode d'Otsu² est appliquée (f). Une liste binaire est obtenue et permet de reconstituer l'entier qui lui correspond afin de trouver quel est l'ArUco correspondant.

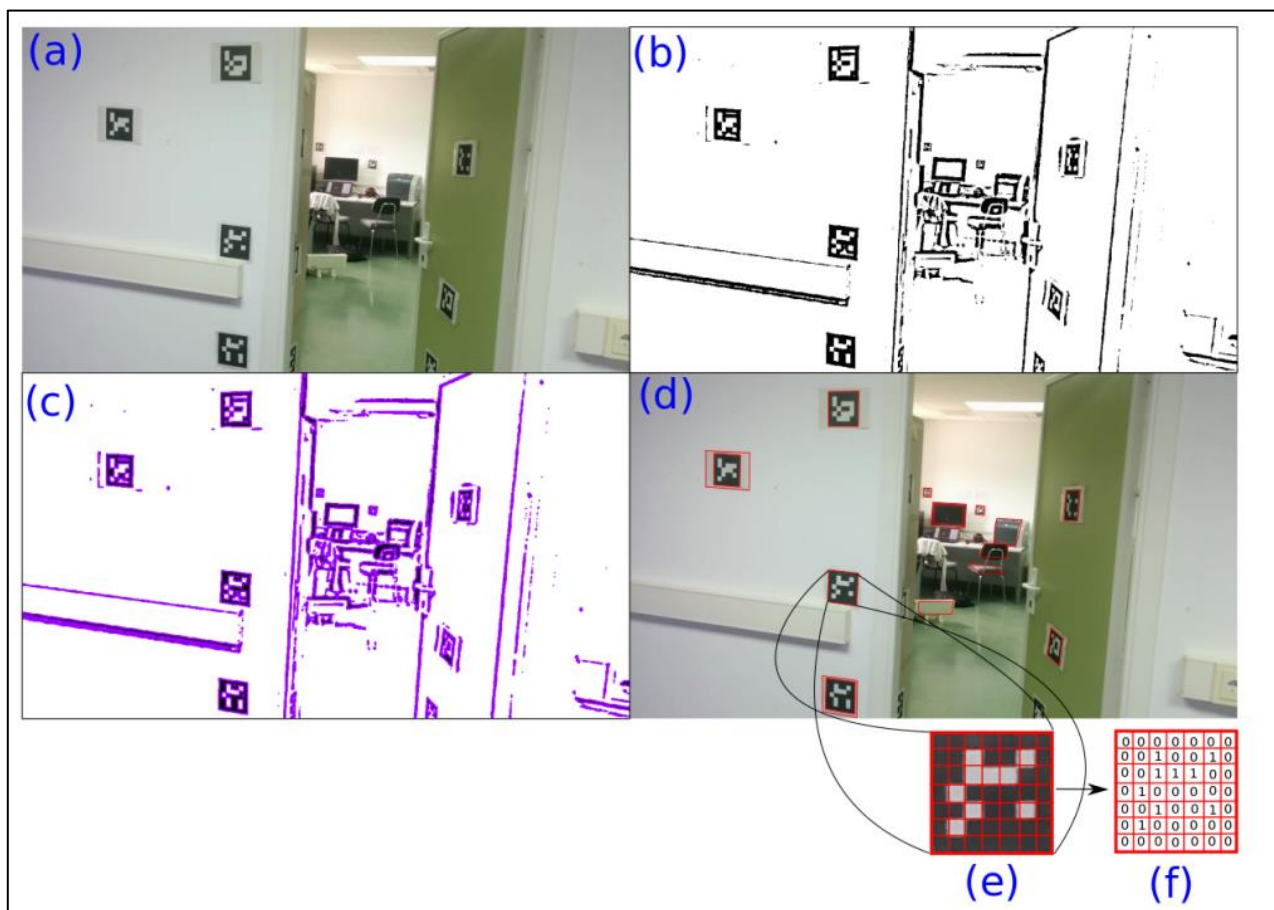


Figure 4 Etapes de traitement d'une image contenant des ArUcos

² https://fr.wikipedia.org/wiki/M%C3%A9thode_d'Otsu

Les processus principaux pour la détection rapide et l'identification sont les suivants :

(Figure 3) L'image originale (a) est redimensionnée pour la recherche des ArUcos. Un seuillage est appliqué (c) afin de trouver les rectangles entourés en rouge (d). Une correspondance avec les marqueurs originaux (e) est faite et accélérée grâce à une Pyramide (f). Les coins des ArUcos sont ensuite utilisés pour trouver leurs localisations dans l'image originale.

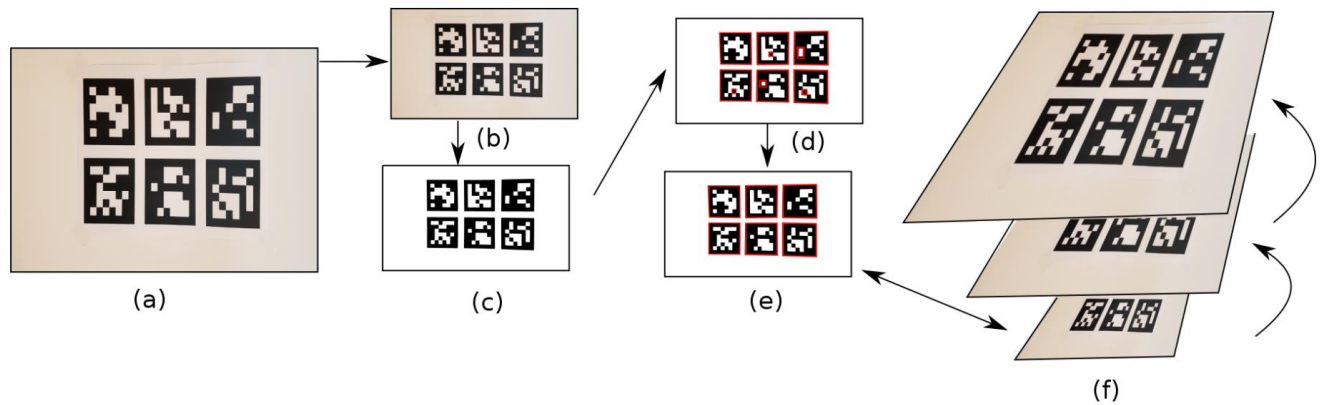


Figure 5 Processus de pipeline

2.3 Interprétation 3d d'une image et calibration d'une caméra

Dans la vision par ordinateur, le modèle pin-hole [4] représente la façon dont une caméra projette une scène 3d vers une image 2d. La vue de la scène est obtenue en projetant un point 3D de la scène dans une image 2d en utilisant la transformation de perspective qui permet d'obtenir le pixel correspondant.

2.3.1 Modèle sans distorsion

Dans la Figure 6, F_c est la position de la caméra dans son système d'axes 3d (X_c, Y_c, Z_c). Le pixel au point (u, v) est présent dans le système d'axes 2d x, y . Le point P est le point correspondant au pixel (u, v) présent dans la scène qui possède son propre système d'axes (X_w, Y_w, Z_w).

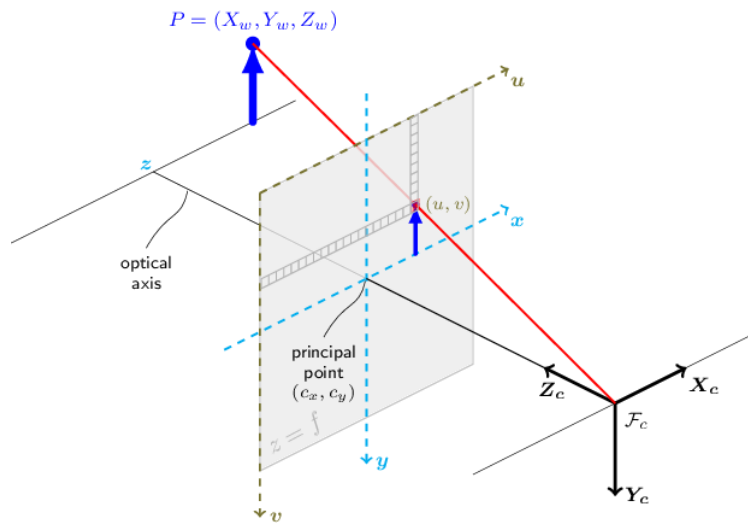


Figure 6 Modèle de caméra pin-hole [4]

La matrice A de la caméra permet de projeter un point de la scène en 3d vers ses coordonnées en pixels dans l'image. Elle prend habituellement la forme suivante (Figure 7) :

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 7 Matrice intrinsèque de la caméra

Une telle projection est montrée dans la Figure 8.

$$\begin{bmatrix} s \cdot u \\ s \cdot v \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

Figure 8 Projection de point 3d vers pixel

Pour trouver le point (u,v) , il suffit de les diviser par s (la coordonnée homogène).

2.3.2 Modèle avec la distorsion

Les objectifs des caméras ont toutes des distorsions dues à la courbure de la lentille. Afin de pouvoir passer d'un point 3d à un point 2d grâce au modèle expliqué au chapitre 2.3.1, il est nécessaire de lui ajouter la prise en compte de la distorsion [4].

La formule suivante est utilisée pour prendre en compte les coefficients de distorsion (Figure 9) :

$$\begin{aligned} & \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x x'' + c_x \\ f_y y'' + c_y \end{bmatrix} \\ \text{where} \\ & \begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + 2p_1 x' y' + p_2 (r^2 + 2x'^2) + s_1 r^2 + s_2 r^4 \\ y' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + p_1 (r^2 + 2y'^2) + 2p_2 x' y' + s_3 r^2 + s_4 r^4 \end{bmatrix} \\ \text{with} \\ & r^2 = x'^2 + y'^2 \\ \text{and} \\ & \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix}, \end{aligned}$$

Figure 9 Formule utilisée pour corriger la distorsion [4]

2.3.3 Calibration de la caméra

Afin d'utiliser une caméra, il est nécessaire de trouver sa matrice intrinsèque ainsi que ses coefficients de distorsions. Pour ce faire, OpenCV fournit un algorithme permettant de détecter ces paramètres à partir de plusieurs photos prises avec la caméra.

2.4 Choix du système embarqué

Afin de réaliser le traitement d'image, et dans une nécessité de performance, des systèmes embarqués ont été comparés puis un d'entre eux a été choisi. Des benchmarks ainsi qu'une comparaison sous forme de tableau multicritère ont été réalisés pour simplifier ce choix.

Le choix a été fait entre les deux systèmes embarqués les plus répandus dans le domaine de la vision par ordinateur :

2.4.1 Jetson Nano

Le Jetson Nano est un système embarqué développé par Nvidia. Il utilise un GPU contenant 128 cœurs CUDA ce qui permet de transmettre une partie de la charge du CPU et ainsi être plus efficace pour effectuer des traitements gourmands. Il peut être alimenté soit par les pins soit par un connecteur micro-USB, soit par l'alimentation DC 5 volts

Il possède un CPU 4 cœurs ARM A57 cadencé à 1.43 GHz et 4GB de ram. Son WI-FI intégré ne gère que les transmissions 2.4 GHz



Figure 10 Jetson Nano (source : <https://developer.nvidia.com/>)

2.4.2 Raspberry Pi 4 B

Le Raspberry Pi 4 B est un système embarqué puissant, réputé et très populaire. Il peut être alimenté soit par les pins, soit par son alimentation USB type-c.

Il possède un CPU 4 cœurs ARM A72 cadencé à 1.52GHz et 4GB de ram. Son Wi-Fi intégré gère les transmission 5.0GHz.



Figure 11 Raspberry pi 4 b (source : <https://www.raspberrypi.org/>)

2.4.3 Benchmarks

Les résultats des benchmarks ont été mis en page dans le tableau ci-dessous afin de les rendre plus lisibles :

	Jetson Nano		Raspberry Pi 4	
Algorithme testé	1x	50x	1x	50x
Prise de vue (Raspberry Pi) / copie de la mémoire (Jetson)	0.011 s	0.547 s	0.762 s	38.107 s
Algorithme de détection des ArUcos présents	0.526 s	26.277 s	0.638 s	31.915 s
BackgroundSubstraction de matrice aléatoire	0.256 s	12.229 s	0.804 s	80.351 s
BackgroundSubstraction de matrice aléatoire (CUDA)	0.053 s	2.668 s	N/A	N/A

Figure 12 Benchmark des systèmes embarqués

La différence dans les tests du temps nécessaire à la prise de vue est due au fait que le Jetson Nano fait par défaut des prises de vues en parallèle alors que le Raspberry Pi 4 (avec la librairie utilisée lors des tests) ne parallélise pas la prise de vue.

Le Jetson Nano ressort gagnant de ce benchmark grâce à son GPU et à sa gestion automatique de la parallélisation des prises des vues.

2.4.4 Critères de choix

Le tableau multicritère ci-dessous a été réalisé pour comparer les aspects externes aux systèmes embarqués.

	Poids	Jetson Nano		Raspberry pi 4	
Performances	5	Benchmark chapitre 2.4.3	5	Benchmark chapitre 2.4.3	2
Simplicité d'utilisation	2	GPU + CUDA	2	CPU	4
Taille (installation)	2	70 x 45 mm	2	85,60 x 53,98 mm	4
Consommation	2	5 volts 2 ampères	3	5 volts 2 ampères	3
Wi-Fi 5 GHz	1	5 GHz avec dongle	2	5 GHz sans dongle	4
Total	12		41		36

Figure 13 Tableau multicritère pour le choix du système embarqué

La simplicité d'utilisation concerne la facilité d'installation de tous les composants nécessaires à l'exécution de l'algorithme final (système embarqué, librairies, drivers, etc...) et à l'utilisation du GPU.

Le choix final s'est porté sur le Jetson Nano.

2.5 Choix de caméra

Le choix de n'utiliser qu'une seule caméra a été fait car une caméra couvrant toute la table a été trouvée et qu'il était directement possible de l'interfacer avec le système embarqué sans avoir besoin de hardware supplémentaire.

Pour la caméra, les nécessités étaient de pouvoir voir toute l'aire de jeu sur chaque prise de vue avec la plus grande résolution possible.

Le choix de la caméra se portait sur deux candidats potentiels :

2.5.1 Arducam imx219 wide angle :

La Arducam imx219 a été conçue pour les systèmes Pi V2. Elle permet de filmer en 4k (8.3 mégapixels) à 30 images par secondes ou en 1080p (2 mégapixels) à 60 images par secondes. Elle utilise un grand angle qui permet d'avoir 175° de champ de vision latéral. Son interface est MIPI CSI-2.



Figure 14 Arducam imx219 (source : <https://www.pi-shop.ch/>)

2.5.2 Raspberry pi camera fisheye :

La Raspberry pi camera fisheye est une caméra conçue pour les systèmes Pi V1. Elle permet des prises de vues en 5 mégapixels et a un angle de vue latéral de 170°. Elle permet de filmer en 1080p (2 mégapixels) à 30 images par secondes.



Figure 15 Raspicam fisheye (source : <https://www.pi-shop.ch/>)

2.5.3 Test de résolution des caméras

Un test de résolution a été réalisé pour les deux caméras Figure 16. Gauche : Arducam imx219. Droite : Raspberry pi cam fisheye.:



Figure 16 Images des caméras. Gauche : Arducam imx219. Droite : Raspberry pi cam fisheye.



Figure 17 Zoom sur les images des caméras. Gauche : Arducam imx219. Droite : Raspberry pi cam fisheye.

Les différences de résolutions sont bien visibles sur la Figure 17. Les différences de traitement des couleurs ne sont pas importantes car l'algorithme fonctionne en noir et blanc.

2.5.4 Critères de choix

Le tableau multicritère suivant a été réalisé pour les caméras :

	Poids	Raspberry pi camera fisheye		Arducam imx219 wide angle	
Résolution	5	2K	3	4K	5
Angle horizontal	2	170°	3	175°	4
Version	1	1	1	2	5
Total			26		38

Figure 18 Tableau multicritère pour le choix de la caméra

La Arducam imx219 wide angle est objectivement meilleure sur tous les points comparés. Elle a donc été choisie pour le projet.

3 Spécification

Le chapitre décrit l'organisation physique des éléments du système, la manière dont ils sont paramétrés et comment ils communiquent.

3.1 Organisation physique

Le système est composé de plusieurs éléments physiques :

- Arducam imx219 wide angle : La caméra du système. Elle est connectée par câble MIPI au Jetson nano et lui transmet les images qu'elle prend.
- Jetson Nano : Il récupère les images de la caméra, trouve les positions des marqueurs ArUco puis envoie les positions aux robots
- Robots alliés : Les deux robots alliés ont chacun un marqueur sur le dessus et 3 sur les côtés du robot. Ils reçoivent les positions que le Jetson Nano transmet.
- Robots ennemis : Les robots ennemis ont un cube avec un marqueur ArUco sur le dessus et 4 marqueurs ArUco sur les bords
- ArUco central : ArUco présent au centre de l'aire de jeu dont la position exacte est connue
- Webtools : Un serveur créé par l'équipe Eurobot RTFM contenant une application web qui contient des outils permettant la surveillance des robots et des systèmes liés aux robots

Le schéma suivant (Figure 19) montre l'organisation des éléments décrits précédemment :

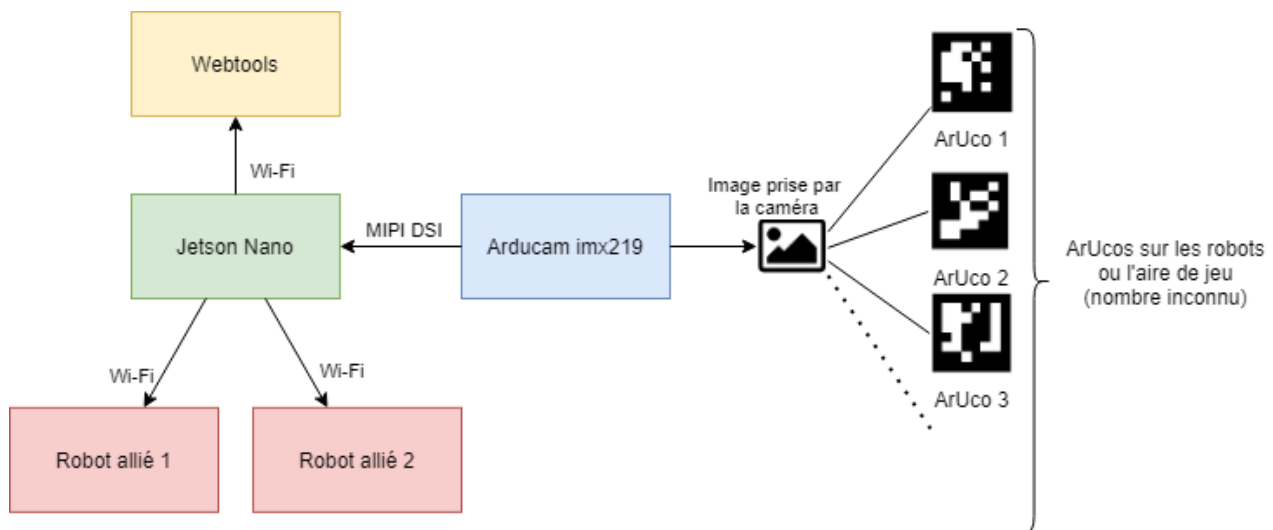


Figure 19 Organisation physique du système

Le Jetson Nano est relié à l'Arducam imx219 via MIPI DSI et envoie les informations aux robots via Wi-Fi.

3.2 Positions spécifiques des ArUcos

Le règlement [1] définit précisément de quelle manière les ArUcos 4x4 peuvent être utilisés. Les ArUcos à disposition changent en fonction de la couleur qui est attribuée à l'équipe Eurobot. Les ArUcos à disposition sont les suivants :

- ArUcos 1 à 5 : Réservés pour être placés au-dessus des robots de l'équipe bleue aléatoirement

- ArUcos 6 à 10 : Réservés pour être placés au-dessus des robots de l'équipe jaune aléatoirement
- ArUcos 11 à 50 : Réservés pour l'aire de jeu et ses éléments
- ArUcos 51 à 70 : Réservés pour l'équipe bleue, ils peuvent être placés sur n'importe quel élément de l'équipe bleue
- ArUcos 71 à 90 : Réservés pour l'équipe jaune, ils peuvent être placés sur n'importe quel élément de l'équipe jaune

En connaissant ces informations, les seules variables qui peuvent changer au début d'une manche sont les ArUcos présent sur le dessus des robots ainsi que la couleur de l'équipe. Avec ces informations, le système peut déduire quels ArUcos seront utilisés pour quels robots.

3.3 Configuration du système

Pour pouvoir modifier le système et l'adapter en fonction de quels ArUcos sont utilisés par l'équipe Eurobot RTMF, des fichiers de configurations sont utilisés

3.3.1 Fichier "aruco_translation.json"

Ce fichier permet d'indiquer quels ArUcos sont utilisés par l'équipe, à quel robot ils se réfèrent et à quelle position relative de celui-ci ils sont placés. Il contient un tableau et un dictionnaire :

- *aruco_list* : liste contenant une entrée pour chaque ArUco utilisé par l'équipe Eurobot RTMF. Chaque entrée est un dictionnaire à 3 clés :
 - *id* : l'id de l'ArUco
 - *robot* : l'id du robot sur lequel l'ArUco est fixé
 - *translation* : le nom de l'endroit où l'ArUco est placé (c'est une référence vers *translation_list*)
- *translation_list* : dictionnaire ayant pour clé le nom de l'endroit où l'ArUco est placé et comme valeur un tableau contenant le vecteur 3d qu'il est nécessaire d'appliquer au système de coordonnées de l'ArUco afin de trouver le point du centre du robot

3.3.2 Fichier "Configuration.py"

Ce fichier contient les configurations des éléments externes au système.

Server

Contient les informations des serveurs auxquels envoyer les positions des robots trouvés par le Jetson Nano et les options spécifiques au serveur.

Camera

Contient les informations de la résolution des prises de vues et du nombre d'images prises par seconde.

Système

Contient les informations suivantes :

- La liste des ArUcos à ne pas prendre en compte lors de la détection
- La position et l'identifiant de l'ArUco présent au milieu de l'air de jeu
- La taille des différents ArUcos utilisés
- La taille de l'échantillon sur lequel se base l'algorithme avant d'envoyer les positions aux robots
- Le nom du fichier Json qui contient les translations pour chaque ArUco (chapitre 3.3.1)
- Les identifiants des différents robots qui seront envoyés en parallèle de leurs positions

3.4 Paramètres du système

Le système prend 5 entiers en paramètres tous obligatoires :

- 1 ou 0 : la couleur attribuée à l'équipe Eurobot RTFM
- Entre 1 compris et 10 compris : l'ArUco présent sur le robot 1
- Entre 1 compris et 10 compris : l'ArUco présent sur le robot 2
- Entre 1 compris et 10 compris : l'ArUco présent sur le robot 3
- Entre 1 compris et 10 compris : l'ArUco présent sur le robot 4

Grâce à ces 5 paramètres, le système peut connaître quels ArUcos ont été attribués à quels robots et ainsi donner les positions correspondantes à chaque robot dont son ArUco aura été trouvé sur l'image.

3.5 Protocole de communication

La communication est implémentée de sorte qu'il soit possible d'envoyer les positions aux robots et aux webtools.

3.5.1 Communication avec les robots

La spécification du protocole de communication permettant d'au Jetson Nano d'envoyer les résultats de son traitement au serveur fonctionnant sur les robots est décrit dans ce chapitre. Il fonctionne par paquets TCP dont la forme est définie dans le code du robot le serveur du robot.

Le serveur écoute sur le port 5000. Il attend la réception d'un paquet TCP. Lorsqu'il en reçoit un, il lit l'identifiant du paquet et s'il correspond à un des identifiant qu'il connaît, assigne les valeurs qui suivent dans le paquet aux valeurs d'un nouvel objet créé. Il est donc nécessaire de transmettre les mêmes types de données que celles qui seront lues.

Les données sont ordonnées comme suivant (

Byte	0	1	2	3	4	5	6	7	8	9
Donnée	id du message	id du robot	position x			position y				

Figure 20) :

Byte	0	1	2	3	4	5	6	7	8	9
Donnée	id du message	id du robot	position x			position y				

Figure 20 Format du paquet TCP envoyé au robot

3.5.2 Communication avec les webtools

La communication avec les webtools fonctionne comme celle décrite au chapitre 3.5.1 sauf pour le port et l'ordre des données. Le port est le 5500 et l'ordre des données est le suivant :

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Donnée	id du msg	couleur R	couleur G	couleur B	nombre de points	position x du point 1	position y du point 1	position x du point 2	position y du point 2					

On peut lui donner autant de points qu'on veut en spécifiant le nombre de points dans le byte 4 et 5.

4 Conception

Ce chapitre décrit comment le projet a été conçu. Les techniques décrites durant ce chapitre sont implémentées dans le chapitre réalisation (chapitre 4.4).

4.1 Fonctionnement du programme

Le diagramme d'activité suivant (Figure 21) décrit quelles seront les étapes par lesquelles va passer le programme.

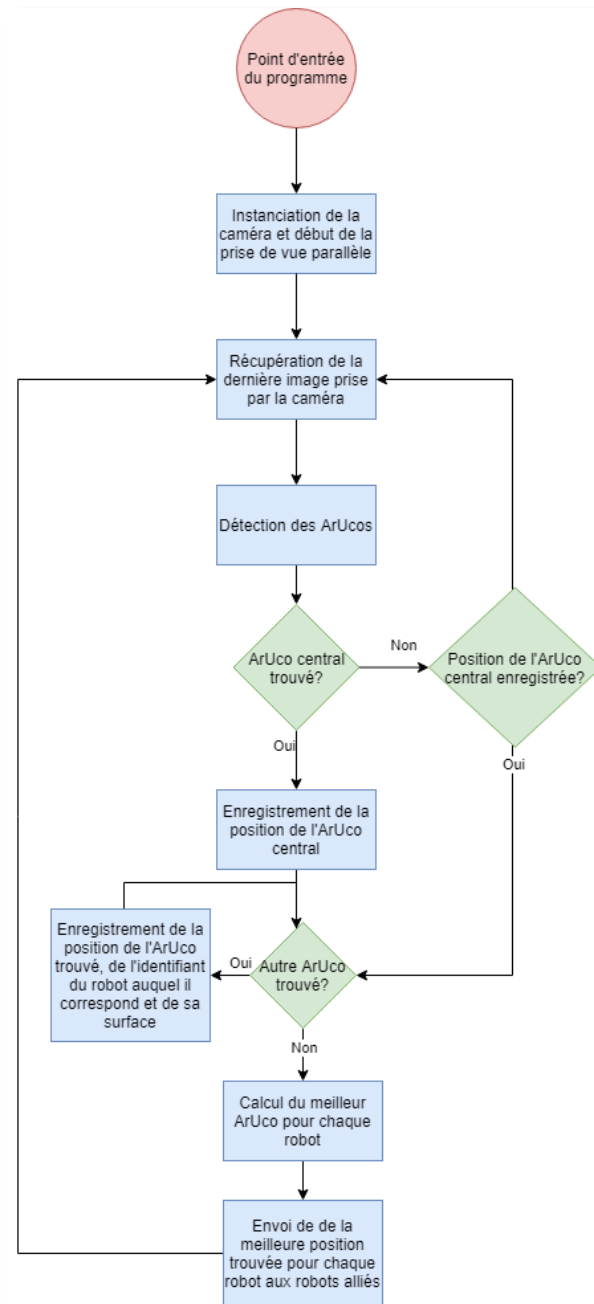


Figure 21 Diagramme d'activité

4.2 Récupération de la position du robot

Ce point décrit quelles sont les étapes à suivre afin de trouver la position d'un robot à partir de l'ArUco central et d'un ArUco détecté à une autre position dans l'image.

La détection des ArUcos est effectuée par la librairie décrite au point 2.2. Les étapes de calcul sont décrites dans les paragraphes suivants. Ils sont détaillés dans [5].

4.2.1 Définition de la matrice de l'aire de jeu

La matrice de l'aire de jeu est une matrice qui définit le point $(0,0,0)$ comme étant le point en haut à droite au niveau 0 de l'aire de jeu. Dans cette matrice, le centre de l'ArUco central est situé au point $(1250,1500,0)$.

Son système d'axe et les points importants sont représentés sur la Figure 22.



Figure 22 Aire de jeu. Le point $0,0,0$ est en haut à gauche de l'aire de jeu et le point $1250,1500,0$ au milieu de l'ArUco central

4.2.2 Récupération de la position de la caméra à partir de l'ArUco central

Le problème d'estimation de la position d'une caméra calibrée à partir d'un set de points 3d et leurs correspondant 2d est connu sous le nom de *perspective-n-point problem* (Figure 23). La position de la caméra est constituée des 6 degrés de liberté : longitude, transversal, vertical, roulis, tangage et lacet. Ce problème possède des implémentation open source qui le résolve pour $n \geq 3$ points. Notre situation utilise un $n=4$. La librairie utilisée [5] possède une implémentation de la résolution de ce problème. Les étapes nécessaires à sa résolution sont décrites dans le prochain paragraphe

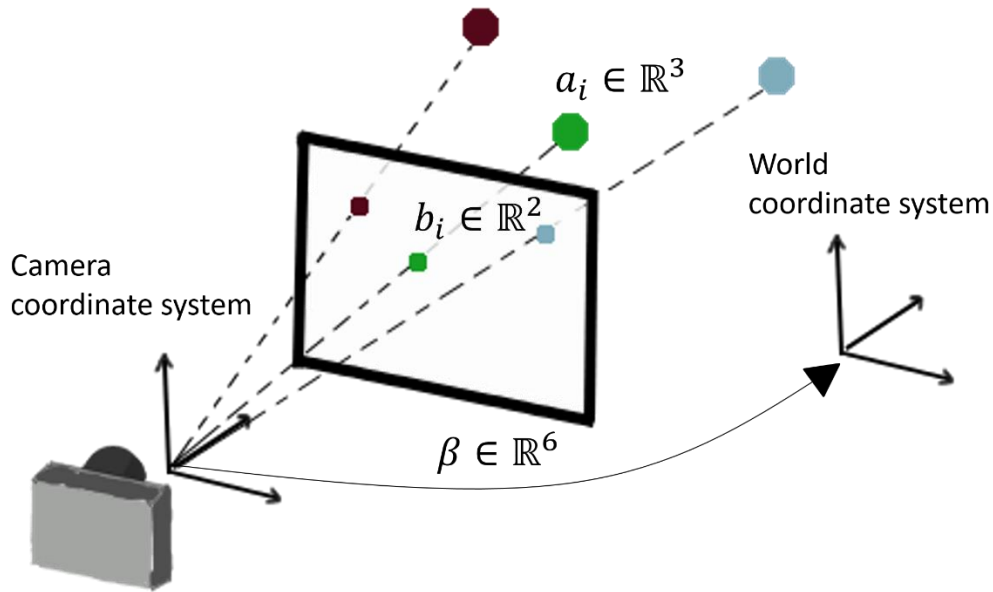


Figure 23 Représentation du perspective-n-point problem

Lorsqu'on connaît la taille du marqueur ainsi que la matrice intrinsèque et les coefficients de distorsion, la librairie permet d'estimer la position relative de la caméra. L'algorithme analysé au chapitre 2.2 nous donne la position en pixel des quatre coins de l'ArUco détecté ainsi que son identifiant. A partir de ces 4 points, nous pouvons trouver une estimation de la position de la caméra dans le système de coordonnées de l'ArUco sous forme de vecteurs translation et rotation. Ils représentent la transformation depuis l'ArUco jusqu'à la caméra.

Grâce aux vecteurs translation et rotation, on peut trouver l'ArUco dans le système de coordonnées de la caméra en faisant une multiplication de matrices. On peut ensuite le multiplier par le système de coordonnées de l'aire de jeu afin d'obtenir la matrice contenant la position de la caméra dans ce système. Il suffit ensuite de diviser la matrice par la coordonnée homogène pour avoir la position X, Y, Z de la caméra.

4.2.3 Récupération de la position du robot à partir d'un ArUco trouvé

Grâce à la position X, Y, Z de la caméra et aux quatre coins d'un ArUco trouvé (Figure 24) on peut, comme au point 4.2.2, récupérer les vecteurs translation et rotation de l'ArUco détecté, les multiplier par la matrice du système de la caméra puis multiplier le résultat par la matrice du système de l'aire de jeu afin d'obtenir la matrice de l'ArUco détecté dans le système de l'aire de jeu. Diviser ce résultat par la coordonnée homogène nous permet d'obtenir la position X, Y, Z de l'ArUco détecté.



Figure 24 Axes des ArUcos trouvés dessinés sur l'image prise par la caméra

4.3 Conception du support

Le support doit permettre de pouvoir fixer la caméra et le Jetson Nano de manière stable et précis. La pièce suivante (Figure 25) a été modélisée. Le trou au centre permet de le fixer grâce à un boulon. La partie supérieure peut accueillir le Jetson Nano et la partie inclinée à 45° permet de fixer la caméra.

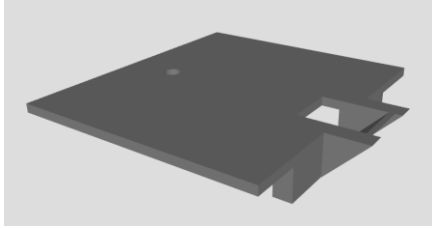


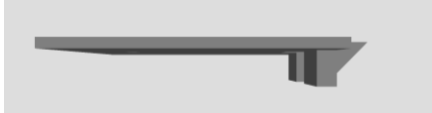
Vue 3d	
Vue de dessus	
Vue de devant	
Vue de côté	

Figure 25 Différentes orientation du support

4.4 Choix du meilleur ArUco par robot

Lors de la détection des ArUcos, il arrive souvent que deux ArUcos soient détectés pour le même robot (Figure 26). Il est nécessaire dans ce cas de sélectionner l'ArUco qui est le mieux visible depuis la caméra afin d'obtenir la meilleure précision (Chapitre 6.2). Afin de déterminer quel ArUco est le mieux visible, un calcul de son aire en pixel est réalisé.



Figure 26 Deux ArUcos sont détectés pour le même robot. Celui de côté a une plus grande aire en pixels que celui du haut

4.5 Utilisation d'un échantillon de données pour augmenter la précision

Afin d'avoir une meilleure précision dans les mesures effectuées par le Jetson Nano, l'utilisation d'un échantillon de données plutôt qu'une donnée seule permet d'utiliser une moyenne ou une médiane pour que la position du robot soit la plus précise possible.

Pour éviter les erreurs dues à une mesure spécifique extrêmement décalée par rapport aux autres, l'utilisation d'une médiane sur l'échantillon est réalisée afin d'obtenir la valeur la plus précise possible.

La taille de l'échantillon est configurable dans le fichier décrit au chapitre 3.3.2.

5 Réalisation

Ce chapitre décrit comment les différents éléments du projet sont réalisés.

5.1 Environnement de développement et logiciels utilisés

Toute l'implémentation du projet a été réalisé sur Visual Studio Code³, un éditeur de code extensible. Le code a ensuite été uploadé sur le Jetson Nano grâce à rsync⁴, un logiciel libre de synchronisation de fichiers. Les transmissions des fichiers de logs (log des position trouvées et log des images enregistrées) ont été faites grâce au logiciel WinSCP⁵. La modélisation du support de la caméra a été réalisée sur Tinkercad⁶.

5.2 Implémentation du programme

Le programme fonctionnant sur le Jetson Nano a été entièrement écrit en python 3. Il gère la prise d'images via la caméra, le traitement de celles-ci afin de trouver les ArUcos et l'envoi des meilleurs positions trouvées aux robots. Il se trouve dans le

5.2.1 Bibliothèques utilisées

Les bibliothèques python3 suivantes ont été utilisés pour réaliser le programme :

Numpy⁷

Bibliothèque destinée à manipuler des matrices et des tableaux à plusieurs dimensions. Contient aussi des fonctions mathématiques opérant sur ces tableaux.

Opencv⁸

Bibliothèque graphique spécialisée dans le traitement d'images en temps réel. Cette bibliothèque possède une implémentation du *perspective-n-point problem* (Chapitre 4.2.2). C'est la bibliothèque qui est utilisée pour récupérer l'image enregistrée par la caméra, détecter la position en pixels des ArUcos, estimer la position de la caméra, calculer une aire à partir de 4 points 2d et pour enregistrer les images prises par la caméra afin d'avoir les images comme log.

Nanocamera⁹

Bibliothèque qui permet d'initialiser le module de caméra depuis le code python et de lui transmettre les paramètres de prise de captures souhaités.

Math

Bibliothèque qui permet d'utiliser des fonctions mathématiques. Elle est utilisée pour effectuer des sinus, cosinus et pour obtenir la valeur approximée de la constante représentant le ratio entre la circonférence d'un cercle et son rayon.

³ <https://code.visualstudio.com/>.

⁴ <https://linux.die.net/man/1/rsync>

⁵ <https://winscp.net/eng/download.php>

⁶ <https://www.tinkercad.com/>

⁷ <https://numpy.org/>

⁸ <https://opencv.org/>

⁹ <https://pypi.org/project/nanocamera/>

Time

Librairie permettant de récupérer le temps en millisecondes de l'horloge du système. Elle est utilisée pour effectuer les tests de performances, calculer le nombre d'images traitées par secondes et calculer la latence du système

Json

Librairie permettant de lire et d'écrire au format Json. Elle permet de lire le fichier décrit au chapitre 3.3.1 afin de pouvoir utiliser ses informations dans le programme.

Sys

Librairie permettant d'utiliser les fonctions du système. Elle est utilisée pour récupérer les paramètres passés en ligne de commande.

5.2.2 Composants

Le programme est divisé en trois modules avec des fonctions différentes.

Server.py

Ce composant contient le fonctionnement du serveur. Il est constitué d'une classe contenant des méthodes statiques. Il permet d'envoyer au robot des paquets composés de :

- Entier entre 1 et 4 : l'identifiant du robot
- Float : La position x
- Float : La position y

Le serveur permet aussi d'envoyer des paquets aux webtools composés de :

- Point : la position x, y
- Color : la couleur du point qu'on désire dessiner
- Id : l'identifiant du point qu'on désire dessiner. Cette information permet de supprimer les anciens points lorsque les webtools en reçoivent un nouveau.

Les webtools sont juste utilisés pour visualiser la position des ArUcos sur le terrain. Ils ne permettent pas de gérer le système du projet ni d'avoir la position précise des Points détectés.

Camera.py

Ce composant contient principalement la matrice intrinsèque de la caméra et ses coefficients de distorsion (Chapitre 2.3.2 **Error! Reference source not found.**). Il contient aussi une méthode statique permettant de récupérer la configuration de la caméra.

Main.py

Ce module, entièrement statique, contient la logique du code. Il est composé de plusieurs méthodes expliqués dans le tableau ci-dessous (Figure 27) :

Nom de la méthode	Paramètres	Valeur(s) de Retour
main()	-	-
Point d'entrée du programme, récupère les paramètres entrés en ligne de commande, instancie les variables nécessaires, instancie la caméra et démarre la prise d'images puis entre dans une boucle infinie dans laquelle il appelle les méthodes qui permettent la détection des ArUcos. Les méthodes appelées sont, dans l'ordre, les méthodes suivantes dans ce tableau. Une fois ces méthodes appelées et les positions des robots trouvés, les positions sont envoyés aux robots puis la boucle recommence.		
checkParams()	params : le tableau contenant les paramètres entrés en ligne de commande	team : la couleur d'équipe assignée 0 pour gauche (bleu) 1 pour droite (jaune) aruco_random_ids : la liste des identifiants des ArUcos posés aléatoirement par les arbitres au début du match dans l'ordre des identifiants des robots
Cette méthode permet d'assigner les paramètres aux variables du programme. Si des mauvais paramètres sont entrés, l'application ne démarra pas et un message d'erreur est affiché.		
getEveryMarkers()	cap : l'objet de capture fournit par opencv aruco_dict : le dictionnaire d'ArUco utilisé (4x4) aruco_params : (les paramètres de la détection)	corners : la liste des coins des ArUcos ids : la liste des identifiants des ArUcos trouvés frame_draw : l'image qui a été utilisée pour trouver les ArUcos
Cette méthode permet de récupérer la dernière image prise par la caméra et d'appeler la fonction OpenCV qui permet de détecter les ArUcos présent sur l'image. Elle rend la liste des ArUcos trouvés avec leurs positions.		
getCentralMarker()	corners : la liste des coins des ArUcos trouvés ids : les identifiants des ArUcos trouvés frame_draw : l'image sur laquelle sont trouvés les ArUcos	is_central_marker_found : boolean vrai si le marker central a été trouvé et faux s'il n'a pas été trouvé transformation_camera_to_world : La transformation qui permet de passer dans les coordonnées de la caméra
Cette méthode permet chercher l'ArUco central. S'il est trouvé, elle calcule la position de la caméra relative à cet ArUco.		
getRobotMarkers()	corners : la liste des coins des ArUcos trouvés ids : les identifiants des ArUcos trouvés frame_draw : l'image sur laquelle sont trouvés les ArUcos team : l'équipe (0 ou 1) définit dans checkParams() aruco_random_ids : liste des identifiants des ArUcos posés	aruco_positions : la liste des ArUcos avec leurs positions x,y,z, le robot auquel il se réfèrent et la surface en pixels définie par les 4 points de l'ArUco

	aléatoirement par les arbitres aux début du match dans l'ordre des identifiants des robots transformation_camera_to_world : matrice qui permet de passer des coordonnées de la caméra aux coordonnées de l'aire de jeu camera_position : la position x,y,z de la caméra aruco_positions : le tableau qui contient chaque ArUco trouvé dans les précédentes détections	
Cette méthode permet de donner la position de chaque ArUco en x,y,z avec le robot auquel l'ArUco se réfère ainsi que la taille de sa surface en pixels.		
getTranslationFromId()	aruco_id : l'identifiant de l'ArUco pour lequel on cherche la translation robot_id : l'identifiant du robot sur lequel il est fixé team : l'équipe (0 ou 1) définit dans checkParams()	translation : tableau de 4 cases qui contient la translation dans les trois axes ainsi que la coordonnée homogène robot_id : l'identifiant du robot qui se réfère à la translation
Cette méthode permet d'aller chercher la translation de l'ArUco dans le fichier décrit au chapitre 3.3.1 à partir de son identifiant.		
getBestPositionsFromArucoList()	aruco_position : liste des ArUcos trouvés dans getRobotMarkers()	best_arucos_by_robot : liste comprenant les meilleurs ArUcos pour chaque robot
Cette méthode permet de trouver quels sont les meilleurs ArUcos trouvés pour chaque robot. Elle utilise l'aire trouvée dans checkParams() pour définir quels ArUcos sont les meilleurs.		
logDetectedAruco()	aruco_position : liste des ArUcos trouvés dans getRobotMarkers() frame_draw : image sur laquelle ont été trouvé les ArUcos	-
Cette méthode permet d'écrire dans un fichier de log les ArUcos trouvés avec leurs positions, leurs surface, l'identifiant du robot auquel ils se réfèrent. Chaque ligne écrite dans ce fichier fait référence à l'image sur laquelle ils ont été trouvés.		

Figure 27 Tableau expliquant les méthodes présentes dans main.py

5.2.3 Génération des logs

Un fichier de logs est écrit en parallèle de la détection des données. Avant chaque envoi des données les logs sont enregistrés dans un répertoire avec la dernière image prise par la caméra compressée.

5.3 Gestion de la déformation de l'objectif de la caméra

Afin d'obtenir la matrice intrinsèque de la caméra ainsi que les coefficients de distorsion des solutions open-source ont été implémentées. Une de ces solutions est présente dans la librairie aruco-3.1.2¹⁰ et permet, à partir de plusieurs images prises avec la caméra, de trouver une approximation de la matrice intrinsèque de la caméra et des coefficients de distorsion. Cet algorithme prend en entrée un répertoire contenant des photos

¹⁰ <https://sourceforge.net/projects/aruco/files/>

puis donne en retour l'approximation de la matrice intrinsèque de la caméra ainsi que de ses coefficients de distorsion.

Il est possible de faire fonctionner cet algorithme avec des ArUcos ou des ChArUcos. Les ChArUcos sont un autre type de marqueurs fonctionnant comme les ArUcos mais avec un damier noir une case sur deux (Figure 28). Ce changement permet d'avoir une détection des coins beaucoup plus précise et donc permet d'approximer les paramètres intrinsèques de la caméra avec plus de précision.

Pour que l'algorithme fonctionne efficacement, il est nécessaire de prendre en photo une board (Figure 28) depuis un maximum d'angle et de distances différents.

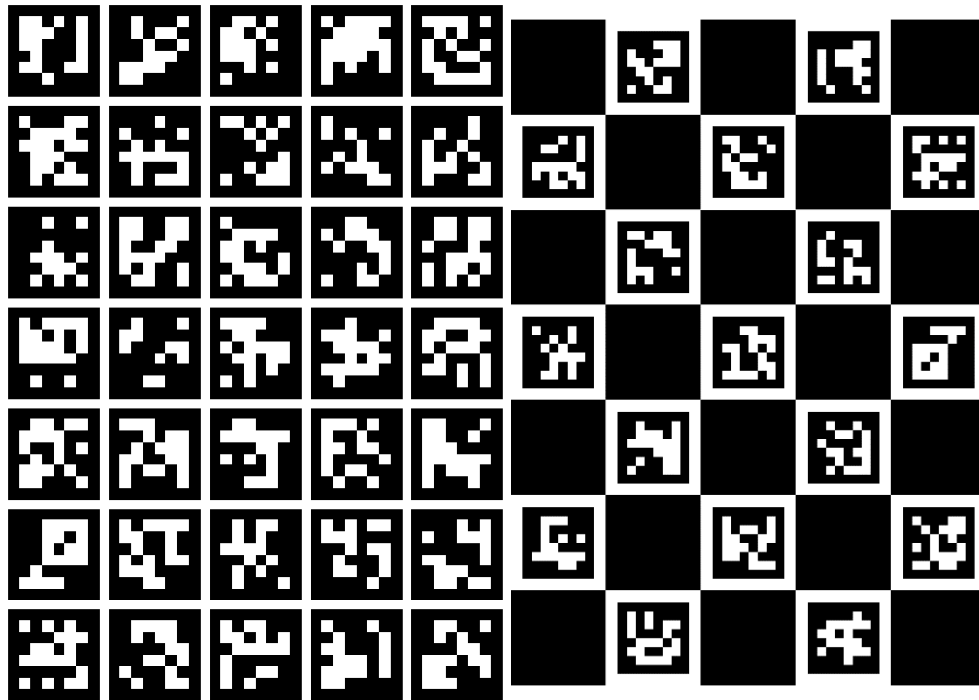


Figure 28 ArUco board gauche vs ChArUco board droite

Au cours du projet, plusieurs matrices ont été réalisées afin d'essayer de trouver celle qui permet d'avoir la meilleure approximation des paramètres intrinsèques de la caméra. La précision obtenue avec la meilleure matrice réalisée sont détaillés dans le chapitre 6.2.

5.4 Réalisation du support

Le support a été imprimé en 3d grâce aux imprimantes 3d fournies par l'équipe Eurobot RTFM. La conception du modèle du support est décrite au chapitre 4.3. Une fois l'objet conçu en 3d, il suffit de le transmettre à l'imprimante 3d pour qu'elle l'imprime.

Le logiciel UP Studio¹¹ a été utilisé pour réaliser l'impression 3d.

5.5 Outils Réalisés et utilisés pendant le projet

Pour simplifier les actions répétitives dans le projet et pour permettre d'avoir des retours visuels, deux scripts externes au projet ont été implémentés.

¹¹ <https://www.tiertime.com/up-studio/>

5.5.1 take_picture.py

Ce script est relativement simple. Il prend une photo avec la caméra et l'enregistre dans un dossier avec un nom. Il permet de simplifier la prise de beaucoup de photos dans le but de les donner à l'algorithme décrit au chapitre 5.3 **Error! Reference source not found.**

5.5.2 undistort.py

Ce script prend une image en entrée (png), applique les coefficients de distorsion défini dans le script à l'image puis rend l'image sans la distorsion de l'objectif (Figure 29). Ce script est très pratique pour permettre d'avoir un retour visuel sur la qualité de la matrice intrinsèque de la caméra obtenue au chapitre 5.3.

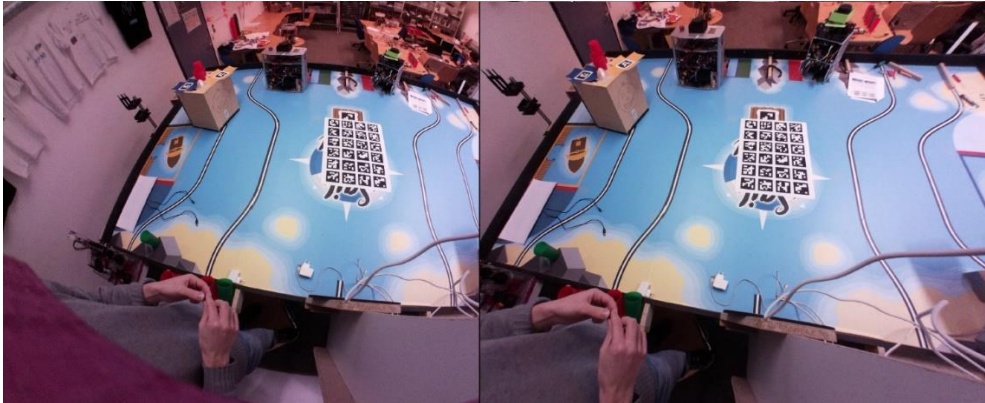


Figure 29 Image de base à gauche image undistort à droite

On peut voir sur l'image de droite (Figure 29) que les lignes de la table sont beaucoup plus droites que celles de l'image de gauche. Cela permet de voir que nos coefficients de distorsion sont relativement bons. Pour prouver qu'ils soient objectivement bons, des tests de précision sont réalisés dans le chapitre 6.2.

5.6 Alimentation du Jetson Nano

Le Jetson Nano peut être alimenté via des GPIO 5 volts et terre. Afin de pouvoir l'alimenter de manière stable, une batterie 12 volts appartenant à l'équipe Eurobot RTFM est utilisée. Un transformateur qui passe le courant 12 volts de la batterie en courant 5 volts a été réalisé par un électronicien de l'équipe Eurobot RTFM. Ce transformateur vient directement se brancher sur les GPIO du Jetson Nano (Figure 30). Cette batterie peut faire fonctionner le Jetson Nano pendant plusieurs heures.

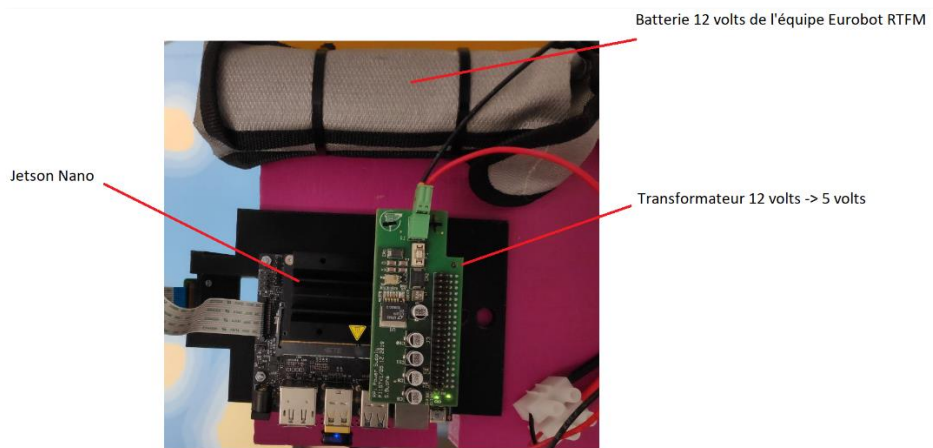


Figure 30 Transformateur réalisé par un membre de l'équipe Eurobot RTFM

6 Tests et validation

Afin de valider le projet, une série de tests ont été effectués. Les tests fonctionnels permettent de s'assurer que le programme se comporte comme attendu dans une situation d'utilisation normale. Les tests de précision et de latence sont faits pour définir à quelle précision on peut s'attendre en utilisant ce système.

6.1 Test fonctionnel

Les tests suivants ont été réalisés (Figure 31). Ils permettent de savoir si les contraintes des objectifs du projet sont respectées.

N°	Description du test	Résultat attendu	Résultat obtenu
1	Le Jetson Nano peut démarrer sur batterie et faire fonctionner l'algorithme pendant au moins 5 minutes	OK	OK
1	Le Jetson Nano est accessible en ssh dès qu'il est allumé et que le routeur de l'équipe Eurobot RTFM est fonctionnel et à moins de 10m	OK	OK
2	Le support peut être installé sur le mât afin que la caméra puisse voir toute l'aire de jeu	OK	OK
3	Le programme démarre en ligne de commande avec les paramètres adéquats	OK	OK
4	Le programme gère correctement les arguments et les fichiers de configuration	OK	OK
5	Groupement correcte des robots possédant les ArUcos du même groupe	OK	OK
6	Le programme donne les positions correctes des robots même avec d'autres ArUcos présents (jamais deux fois le même ArUco)	OK	OK
7	Les données de la position des robots détectés sont équivalentes à celles communiquées au robot	OK	OK
8	Les données sont envoyées aux deux robots en parallèle	OK	OK
9	La communication ne bloque pas le programme si le serveur du robot n'est pas disponible	OK	OK

Figure 31 Tests fonctionnels

6.2 Précision et latence

Le but du projet était de pouvoir trouver la position des robots à 5cm près en moins de 1 seconde et à 2cm près en moins de 5 secondes. Ces deux objectifs et les résultats du système développé durant ce projet sont décrits dans les paragraphes suivants grâce à différents graphiques.

6.2.1 Résultats des tests pour une détection en moins d'une seconde

Le but de ces tests de précision est de mesurer quelle est la précision lorsqu'on est limité dans le temps. Ce chapitre explique quelle précision donne l'algorithme pour des mesures qui prennent moins d'une seconde.

Le box-plot suivant (Figure 32) donne une représentation du temps nécessaire à l'algorithme pour effectuer la détection des ArUcos, les calculs nécessaires pour trouver la position des robots présents sur l'aire de jeu

et l'envoi des informations à ces robots. La valeur maximale sur les 112 mesures de temps effectuées est 510 millisecondes. Le temps nécessaire à la prise d'une mesure est d'environ 370 millisecondes.

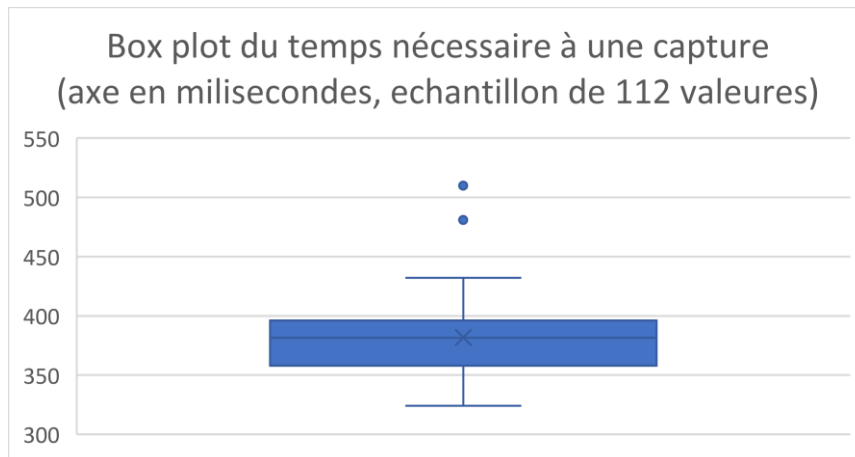


Figure 32 Box plot des temps nécessaires l'analyse d'une image et la communication des résultats au robot

Les graphiques suivants montrent quelle est l'erreur mesurée dans une coordonnée par rapport à la position détectée sur l'aire de jeu. L'axe des x (de 500 à 1500) correspond à la position en x dans le système d'axes de l'aire de jeu. L'axe des y (de 500 à 2500) correspond à la position en y dans le système d'axes de l'aire de jeu. L'axe des z (de 0 à 50) est l'erreur mesurée dans la coordonnée du graphique. Le graphique Figure 33 Erreur en x sur une mesure (valeurs en millimètres) Figure 33 concerne l'erreur en x. Les graphiques suivants (Figure 33, Figure 34 et Figure 35) démontrent respectivement l'erreur en x, en y et en z.

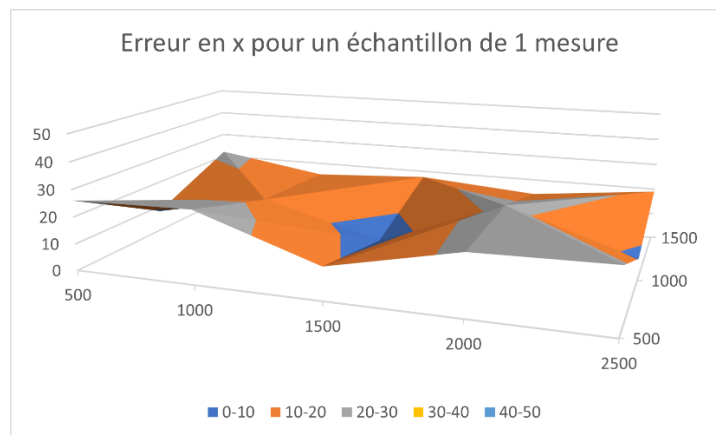


Figure 33 Erreur en x sur une mesure (valeurs en millimètres)

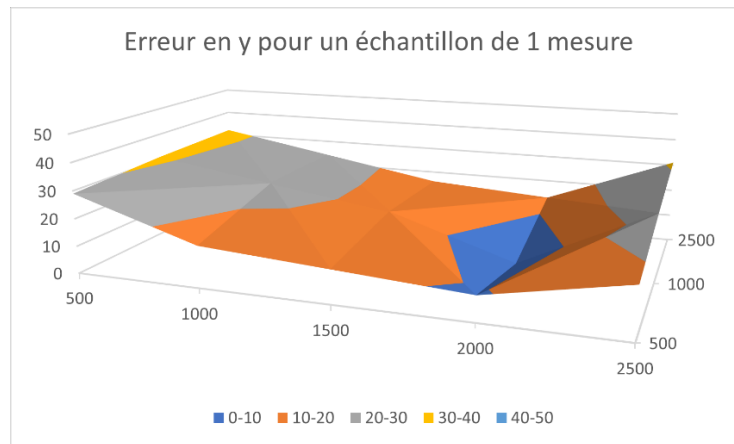


Figure 34 Erreur en y sur une mesure (valeurs en millimètres)

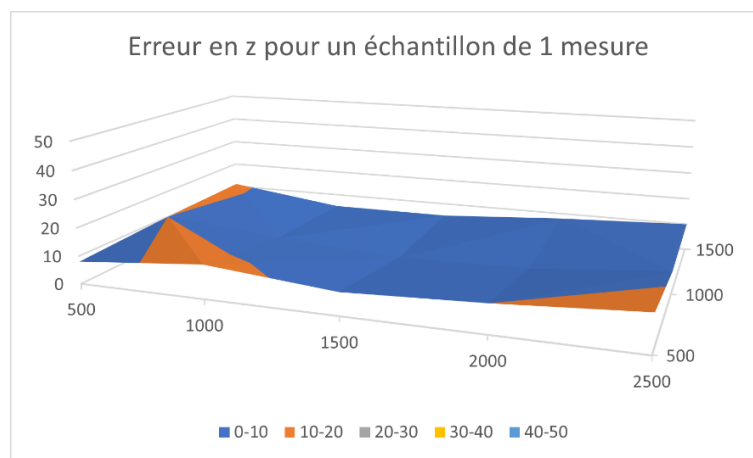


Figure 35 Erreur en z sur une mesure (valeurs en millimètres)

L'erreur maximale présente sur ces graphiques est de 35mm lorsque le robot est à la position (1500, 500). L'erreur moyenne est de 17.45mm et l'erreur médiane est de 13mm.

6.2.2 Résultats des tests pour une détection en moins de 5 secondes

Dans ce chapitre, plusieurs mesures sont réalisées à la suite sur plusieurs images prises afin de profiter des 5 secondes à disposition. Les graphiques ci-dessous (Figure 36, Figure 37, Figure 38 et Figure 39) suivent la même logique que ceux expliqués au point précédent (Chapitre 6.2.1). Ils ont été réalisés lorsque l'algorithme prend 10 mesures avant d'envoyer la position qu'il estime être la plus cohérente. Le temps nécessaire à la prise de 10 mesures est d'environ 3000 ms.

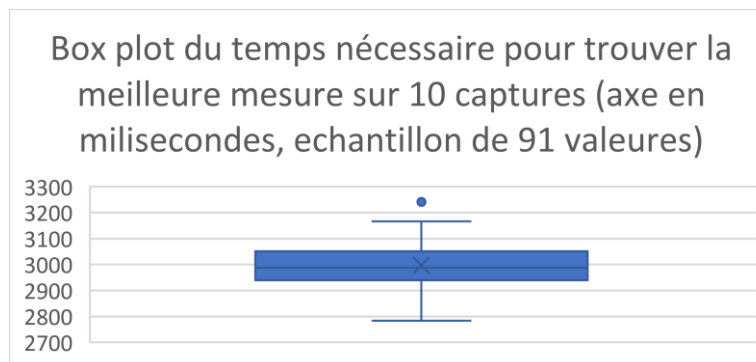


Figure 36 Temps nécessaire au système pour trouver la meilleure valeur sur 10 images analysées

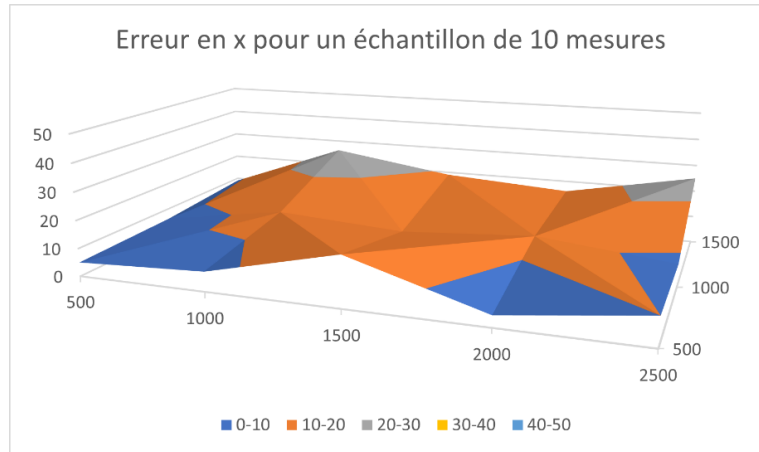


Figure 37 Erreur en x sur 10 mesures

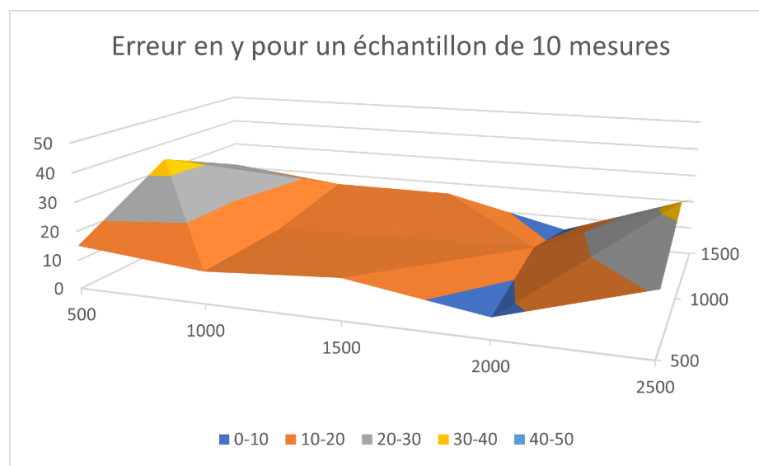


Figure 38 Erreur en y sur 10 mesures

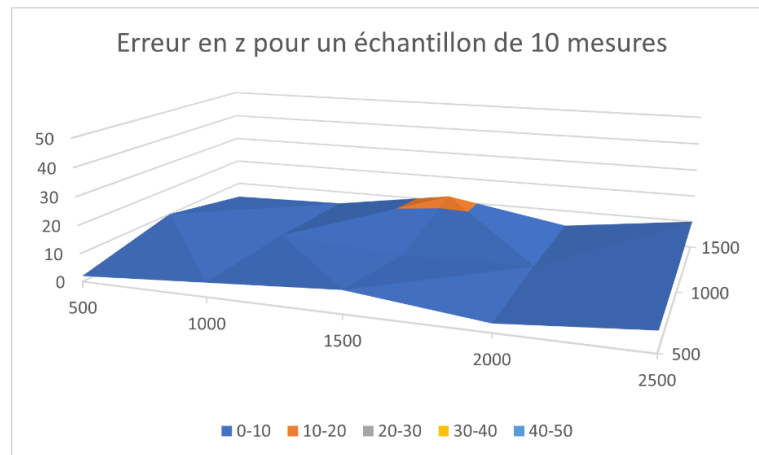


Figure 39 Erreur en z sur 10 mesures

L'erreur maximale présente sur ces graphiques est de 36mm lorsque le robot est à la position (1000, 500). L'erreur moyenne est de 14.925mm et l'erreur médiane est de 13mm. Le fait de prendre plusieurs photos afin d'estimer la meilleure position par rapport à toutes les images traitées a permis de faire baisser la moyenne de 17.45 (une mesure) à 14.925 (10 mesures). La médiane reste inchangée. La déduction de ces résultats permet d'aboutir à l'hypothèse suivante : le fait de prendre un échantillon de plusieurs valeurs permet de lisser les valeurs transmises au robot.

Grâce à ce lissage, le système peut transmettre aux robots des valeurs en lesquelles il a davantage confiance. Cette amélioration de la précision moyenne a cependant un cout : il a fallu environ 10 fois plus de temps au système pour donner la position des robots que lorsqu'il a traité une seule image.

7 Améliorations et travaux futurs

Le projet a été principalement constitué de recherche de solutions pour effectuer une tâche précise. Il y a beaucoup de paramètres à prendre en compte pour réaliser un système comme celui qui a été réalisé dans ce projet. Il y a aussi beaucoup d'alternatives pour trouver la position d'un marqueur filmé. En plus de ça, il y a énormément de matériel différent envisageable. Ce chapitre décrit les améliorations ou modifications qui n'ont pas eu la chance d'être implémentées dans le projet final.

7.1 Choix de l'algorithme de détection des marqueurs ArUco

Plusieurs algorithmes de détection des marqueurs ArUco sont disponibles en open source. Dans ce projet, l'implémentation a été réalisée grâce à la bibliothèque de OpenCV. Une autre bibliothèque envisageable est la bibliothèque ArUco générique (la même qui a été utilisée dans le chapitre 5.3). Cette bibliothèque C++ permet l'utilisation d'options plus poussées et d'autres méthodes de détections. Les appels à cette bibliothèque sont disponibles depuis python3 mais la bibliothèque qui les gère n'est plus mise à jour. Il n'est pas possible de dire si cette modification apporterait une amélioration positive ou non.

7.2 Choix de la caméra

Plusieurs caméras sont disponibles sur le marché. Après plusieurs tentatives pour obtenir la matrice intrinsèque la plus proche de la réalité et les coefficients de distorsion les plus précis possibles, la conclusion suivante a été tirée : il est beaucoup plus facile de calibrer une caméra avec une déformation légère de l'objectif plutôt qu'une caméra grand angle. Plus la déformation est grande, plus la calibration est difficile à approximer.

Le choix vers une caméra avec une courbure de l'objectif moins prononcée permettrait de trouver les matrices intrinsèques et les coefficients de distorsion avec beaucoup plus de précision.

7.3 Utilisation de plusieurs caméras

Lors de l'utilisation d'une caméra avec une courbure de l'objectif faible, il est plus difficile d'observer toute l'aire de jeu à partir du pilier central. Une solution serait l'utilisation de deux caméras en parallèle. Elles pourraient avoir chacune leurs visions sur l'ArUco central et un des côtés de l'aire de jeu. En utilisant ça, on gagnerait l'avantage de ne pas avoir besoin d'utiliser un grand angle ainsi que la possibilité de voir l'aire de jeu en entier. Cela permettrait de trouver les matrices intrinsèques et les coefficients de distorsion beaucoup plus précis et ainsi augmenter la précision totale du système.

7.4 Amélioration des performances

Le programme est actuellement codé en python3 et n'est pas du tout parallélisé. Le GPU du Jetson Nano est utilisé pour générer les images mais pas pour effectuer les calculs de l'algorithme de détection des ArUco. La bibliothèque C++ ArUco¹² 3.1.12 permet d'utiliser les cœurs CUDA du GPU du Jetson Nano pour permettre un traitement de l'image beaucoup plus efficace.

¹² <https://sourceforge.net/projects/aruco/files/>

Il n'existe pas actuellement de librairie qui gère les appels à la librairie C++ ArUco 3.1.12 en python3. Afin d'utiliser cette librairie, plusieurs choix sont alors envisageables :

- Implémenter des appels de méthode à la librairie C++ depuis python
- Implémenter tout le système réalisé dans ce projet en C++
- Implémenter la librairie ArUco en python

Bien que le dernier point décrit ci-dessus soit relativement complexe, les deux autres points sont largement réalisables. Ils permettraient au système d'être beaucoup plus rapide pour traiter les images prises par la caméra.

7.5 Changement des marqueurs présent sur les robots

Pour permettre à l'équipe Eurobot RTFM de changer les marqueurs plus facilement sur les robots, utiliser des écrans qui affichent les marqueurs ArUco pour qu'ils soient changés automatiquement au début d'une manche en fonction de l'attribution de la couleur à l'équipe Eurobot RTFM

7.6 Autres méthodes de détection

D'autres méthodes de détections pourraient être utilisé pour trouver la position du robot. Lors de ce projet, seuls les ArUco ont été envisagés car c'est le seul type de marqueurs qui est réservé pour l'utilisation d'une équipe et donc qui évite toute confusion s'il arrivait qu'une équipe ennemie utilise le même type de détection de position que l'équipe Eurobot RTFM.

7.6.1 Méthodes basées sur l'utilisation d'une caméra

Les idées d'amélioration suivantes se basent sur l'utilisation d'une caméra comme méthode de détection de position.

Utilisation d'autres marqueurs

Les ArUco ne sont pas les seuls marqueurs disponibles pour réaliser une détection de position. Il y a beaucoup d'autres types de marqueurs. Un set qui ressort souvent sont les ChArUco. Ils sont conçus comme les ArUco magiques mais possèdent un damier noir autour d'eux-mêmes (Figure 40)Figure 40 ChArUco board. Ces marqueurs peuvent être plus efficaces car grâce au damier noir, il est plus facile de trouver les coins précis qui correspondent au marqueur et donc avoir une détection des ennemis plus précise.

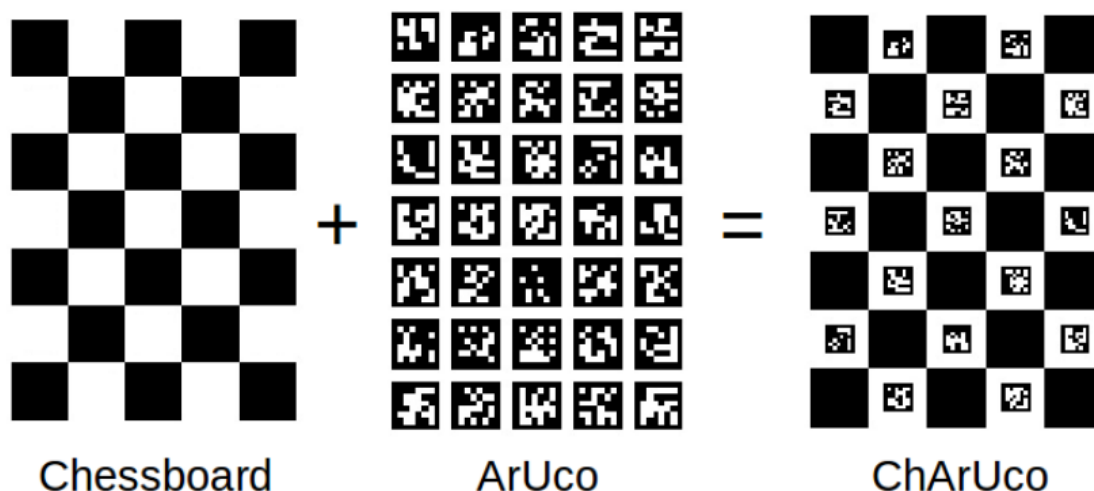


Figure 40 ChArUco board

Utilisation de couleurs, formes, symboles ou lumière

L'utilisation de couleurs, formes ou symboles personnalisés pourrait permettre d'avoir des points de référence plus précis car il n'y a plus de limitation de l'aire sur laquelle la référence est faite. Une solution pourrait être, par exemple, d'utiliser des bandes de couleurs tout autour de la coque du robot. Un système personnalisé à ce point serait cependant nettement plus complexe à implémenter.

7.6.2 Méthodes basées sur l'utilisation d'une caméra en parallèle d'un autre système

Afin d'avoir plus d'information que juste celles générée par la caméra, un couplage à un autre système pourrait être une solution relativement simple à implémenter. Utiliser une technologie comme la Kinect¹³ qui crée un maillage de la distance à laquelle sont les objets qu'elle filme permettrait de connaître une valeur supplémentaire. Grâce à l'ajout de la distance, la position d'un robot estimée par l'algorithme pourrait être beaucoup plus

¹³ <https://en.wikipedia.org/wiki/Kinect>

8 Mise en place et exemple d'utilisation

Ce chapitre décrit comment mettre en place et utiliser le Jetson Nano. Ce chapitre sert aussi de mode d'emploi pour un utilisateur externe qui désirerait utiliser ce système.

8.1 Mise en place et installation du matériel

Ce chapitre décrit les étapes nécessaires afin de pouvoir mettre en place l'environnement du Jetson Nano et de pouvoir l'installer physiquement sur le support prévu à cet effet.

8.1.1 Mise en place de l'environnement du Jetson Nano

Pour pouvoir utiliser le script python écrit durant ce projet, il est nécessaire que python 3 ainsi que les librairies listées au chapitre 5.2.1 soient toutes installées sur le Jetson Nano. Il est ensuite nécessaire que les 3 composant python listés au chapitre 5.2.2 ainsi que les deux fichiers de configuration listés au chapitre 3.3 soient présent sur le Jetson Nano dans le même répertoire. Ces fichiers sont disponibles dans l'annexe de ce rapport. Pour que le Jetson Nano puisse communiquer avec les robots, il faut qu'ils soient atteignables dans le même sous-réseau.

8.1.2 Installation du matériel

Pour installer le matériel, il est nécessaire que le Jetson nano et la caméra soient connectés entre eux grâce aux connecteurs et à un câble MIPI. Le transformateur doit être fixé sur le Jetson Nano comme sur la Figure 41. L'alimentation du transformateur doit être relié à l'alimentation de la batterie. Le Jetson Nano et la caméra doivent être visés au support.

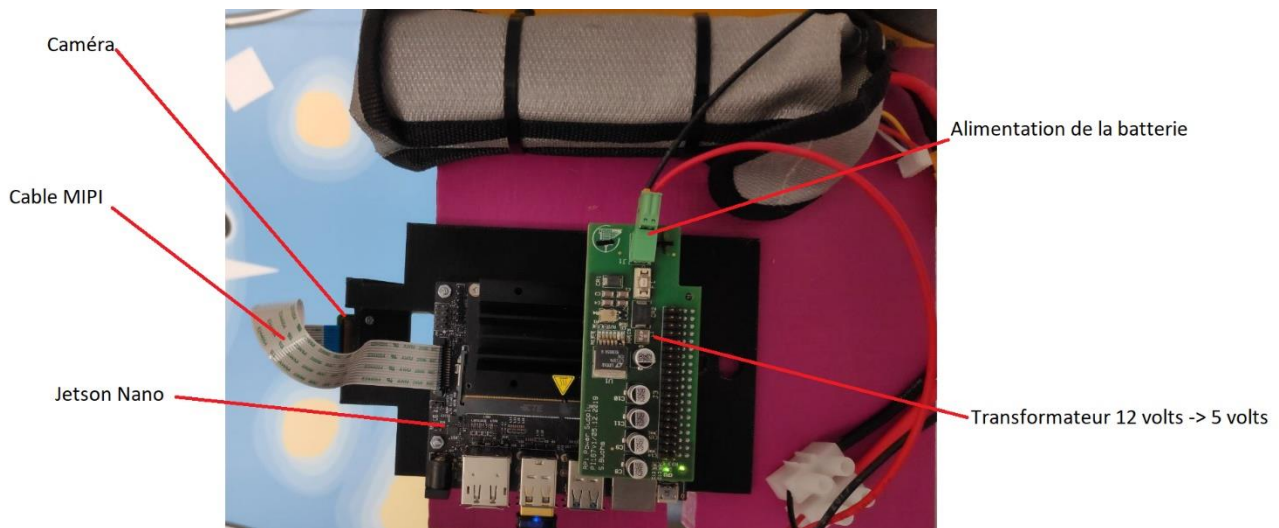


Figure 41 Installation du système vu de dessus

Il faut ensuite poser le support sur le pilier central afin que le coté plongeant du support de la caméra soit plaquée contre le bord comme sur la Figure 42. Une fois le système en place, le brancher à la batterie permet de directement l'allumer.

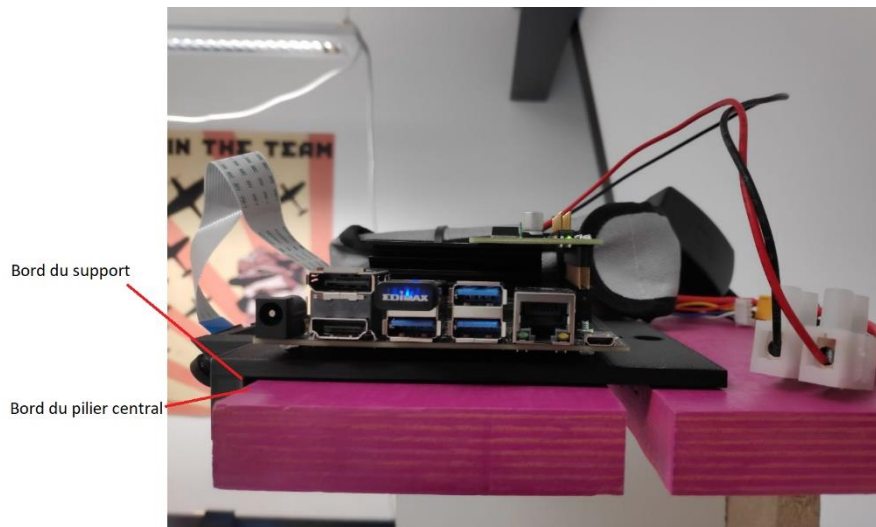


Figure 42 Installation du Jetson Nano vue de côté

Une fois le système installé et allumé, il faut lancer le programme python depuis la ligne de commande.

8.2 Utilisation du programme

Pour utiliser le programme, une fois les chapitres 8.1.1 et 8.1.2 suivis, il suffit de le lancer avec la commande suivante : `python3 main.py arg1 arg2 arg3 arg4 arg5`

Les paramètres `arg1`, `arg2`, `arg3`, `arg4` et `arg5` doivent être remplacés par des valeurs décrites dans le chapitre 3.4. Cette étape, ainsi que son résultat espéré est démontré sur la Figure 43.

```
denis@sauron:~/workspace$ python3 main.py 0 1 5 6 9
GST_ARGUS: Creating output stream
CONSUMER: Waiting until producer is connected...
GST_ARGUS: Available Sensor modes :
GST_ARGUS: 3264 x 2464 FR = 21.000000 fps Duration = 47619048 ; Analog Gain range min 1.000000, max 10.625000; Exposure Range min 13000, max 683709000;
GST_ARGUS: 3264 x 1848 FR = 28.000001 fps Duration = 35714284 ; Analog Gain range min 1.000000, max 10.625000; Exposure Range min 13000, max 683709000;
GST_ARGUS: 1920 x 1080 FR = 29.999999 fps Duration = 33333334 ; Analog Gain range min 1.000000, max 10.625000; Exposure Range min 13000, max 683709000;
GST_ARGUS: 1280 x 720 FR = 59.999999 fps Duration = 16666667 ; Analog Gain range min 1.000000, max 10.625000; Exposure Range min 13000, max 683709000;
GST_ARGUS: 1280 x 720 FR = 120.000005 fps Duration = 8333333 ; Analog Gain range min 1.000000, max 10.625000; Exposure Range min 13000, max 683709000;
GST_ARGUS: Running with following settings:
  Camera index = 0
  Camera mode = 0
  Output Stream W = 3264 H = 2464
  seconds to Run = 0
  Frame Rate = 21.000000
GST_ARGUS: Setup Complete, Starting captures for 0 seconds
GST_ARGUS: Starting repeat capture requests.
CONSUMER: Producer has connected; continuing.
[ WARN:0] global /home/denis/opencv/modules/videoio/src/cap_gstreamer.cpp (933) open OpenCV | GStreamer warning: Cannot query video position: status=0, value=-1, duration=-1
#### new frame
robot : 3
x : 1097.371296270813
y : 1812.0345374189574
z : 349.91447002050324
512
robot : 1
x : 1450.65875878652
y : 788.6852776111784
z : 158.2105836345258
3
#### new frame
robot : 3
```

Figure 43 démarrage du programme avec python3

8.3 Résultat observé sur les robots

On peut observer sur l'image suivante (Figure 44) le résultat qui est affiché dans les logs des robots lorsqu'ils reçoivent une position de la part du Jetson Nano. Chaque ligne correspond à une position d'un robot détectée reçue par le robot allié.

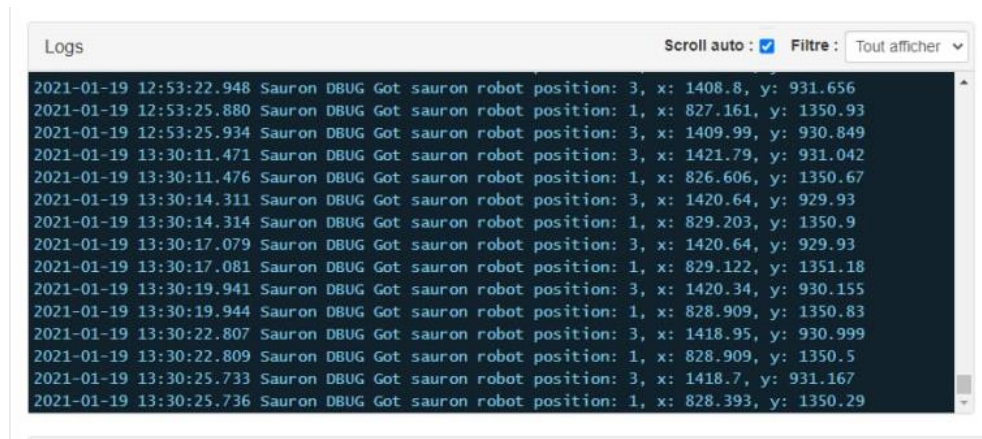


Figure 44 logs des robots (à droite) et résultat de la console du système (à gauche)

Si le robot affiche la position dans son système de logs, c'est qu'il peut l'interpréter et l'utiliser. Le programme va continuer à fonctionner dans la console du Jetson Nano jusqu'à ce qu'il soit interrompu de force (en utilisant le raccourcis CTRL+C par exemple)

9 Conclusion

9.1 Comparaison entre les résultats et les objectifs

Dans ce chapitre, les résultats obtenus dans le chapitre 6 sont confrontés aux objectifs du cahier des charges. Le système répond aux objectifs ainsi qu'aux contraintes imposées par le règlement. Il répond aussi presque à toutes les contraintes imposées par l'équipe Eurobot RTFM. Les contraintes auxquelles il ne répond pas totalement sont les suivantes :

- Les informations à envoyer sont la position de chaque robot en coordonnées x/y en millimètres par rapport à l'aire de jeu ainsi que l'orientation de chaque robot
- Le système doit pouvoir permettre une détection lente et précise des robots avec une précision de moins de 2cm et une latence de moins de 5000ms

Pour la première contrainte, il a été décidé, (après discussion avec l'équipe RTFM) que l'orientation des robots n'était pas utile lors de la détection des robots adverses car connaître cette information ne nous donne pas d'information supplémentaire utile. Pour l'orientation des robots alliés, l'orientation de l'ArUco posé sur son sommet n'est pas contrôlable (vu que c'est l'arbitre qui décide quel ArUco est mis sur nos robots). Il faudrait ajouter des paramètres supplémentaires au programme, ce qui rendrait son utilisation plus complexe. L'équipe Eurobot RTFM a ajouté que l'orientation des robots alliés a toujours été précise et qu'amener une information différente de l'orientation actuelle du robot peut mener à des contradictions dans les données que le robot utilise.

Pour la seconde contrainte, les tests ont démontré (chapitre 6.2) qu'une mesure qui prenait moins de 5 secondes permettait d'arriver à une moyenne de précision de 14.925mm et une erreur médiane de 13mm. Cependant, il arrive que des mesures dépassent de plus d'un centimètre la limite imposée par l'équipe Eurobot RTFM. Afin de respecter cette contrainte, les points expliqués au chapitre 7 peuvent être implémentés dans le futur. Le projet actuel ne respecte donc pas cette contrainte. Après discussion avec l'équipe, il a été admis que cette contrainte était très ambitieuse. Elle reste cependant une contrainte pas respectée.

9.2 Rétrospective sur la planification initiale

La planification initiale a permis de définir sur quels points les heures du projet devaient être attribués. Elle était prévue pour fonctionner en *"agile"* et a permis de revenir sur des choix qui avaient été faits précédemment afin de les modifier et de les améliorer. Elle a permis de souvent revenir sur la manière dont étaient détectés les ArUcos afin de pouvoir itérer sur les solutions qui fonctionnaient le mieux. La planification a été suivie mais certaines parties ont pris plus de temps que ce qui était prévu alors que d'autres ont été réalisées plus rapidement.

Le projet a pu être réalisé dans les délais impartis et les objectifs qu'il remplit ainsi que les contraintes qu'il respecte permettent à l'équipe Eurobot RTFM de l'utiliser pour ses prochains concours.

9.3 Conclusion personnelle

Ce projet a été le projet le plus intéressant que j'ai réalisé dans ma jeune carrière de développeur. Le fait de lier le concours Eurobot avec un projet de semestre a permis d'énormément me motiver à réaliser ce projet. Les domaines liés dans ce projet (systèmes embarqués, robotique, vision par ordinateur) sont parmi les domaines que je préfère et bien que le projet ait été compliqué à réaliser, il m'a permis d'apprendre et de

comprendre beaucoup de concepts intéressants. Le résultat de la solution que j'ai pu développer pour ce projet me satisfait et me motive à continuer à réaliser des systèmes dans ce domaine.

J'aimerais remercier les superviseurs du projet, Monsieur Jacques Supcik et Monsieur Nicolas Schroeter pour leurs conseils et leurs accompagnements tout au long du projet. Leurs expériences ont pu m'aider à mieux me diriger durant le projet et leurs connaissances m'ont permis de trouver beaucoup de réponse aux interrogations que j'ai eu pendant cette réalisation.

10 Déclaration sur l'honneur

Je, soussigné, Denis Rosset, déclare sur l'honneur que le travail rendu est le fruit d'un travail personnel. Je certifie ne pas avoir eu recours au plagiat ou à toute autre forme de fraude. Toutes les sources d'information utilisées et les citations d'auteur ont été clairement mentionnées.

A handwritten signature in black ink, consisting of stylized initials 'DR' followed by a long, sweeping horizontal stroke.

Annexes

Tous les fichiers concernant le projet sont disponibles sur le git de la HEIA-FR à l'adresse :

<https://gitlab.forge.hefr.ch/denis.rosset/sauron2021/-/tree/master/>

Les fichiers sont dans les répertoires suivants :

- **doc** : contient tous les fichiers qui concernent la documentation
 - **cahier des charges** : contient les différentes versions du cahier des charges
 - **defense_cahier_des_charges** : contient la présentation utilisée lors de la défense du cahier des charges
 - **documentation** : contient le rapport final
 - **matrice de risques** : contient la matrice de risque réalisée dans pour le chapitre 2.1
 - **precision_tests** : contient les logs de tous les tests de précision qui ont été effectués au cours du projet
 - **pvs** : contient la liste des procès-verbaux qui ont été rédigés pendant chaque séance
 - **resultats_benchmark** : contient l'output des scripts de benchmark utilisés pour le chapitre 2.4.3
- **librairies** : contient les librairies utilisées pendant le projet
 - **x86 / arm** : versions compilées pour l'architecture x86 ou l'architecture arm
 - **aruco-3.1.12** : librairie utilisée pour générer les valeurs intrinsèques de la caméra
 - **opencv**
- **src** : contient le code du projet sous forme de fichiers sources interprétables
- **utilities** : contient tous les scripts qui ont été réalisés pour simplifier le projet
 - **jetson** : contient les scripts qui ont été développés spécifiquement pour le Jetson Nano ainsi que les scripts de benchmarking
 - **raspi** : contient les scripts qui ont été développés spécifiquement pour le Raspberry Pi ainsi que les scripts de benchmarking

Les modifications apportées au code du robot afin de prendre en compte les messages TCP envoyés par le système réalisé dans ce projet sont disponibles à l'adresse suivante :

https://gitlab.forge.hefr.ch/denis.rosset/sauron2021/-/blob/master/robot_git_diff

Bibliography

- [1] P. Sciences, «Règlement EurobotOpen 2021,» 26 01 2021. [En ligne]. Available: https://www.eurobot.org/images/2021/E2021_Rules_FR.pdf.
- [2] R. M.-S. R. M.-C. Francisco J. Romero-Ramirez, «Speeded Up Detection of Squared Fiducial Markers,» June 2018. [En ligne]. Available: https://www.researchgate.net/publication/325787310_Speeded_Up_Detection_of_Squared_Fiducial_Markers. [Accès le 26 01 2021].
- [3] R. M.-S. F. J. M.-C. R. M.-C. S. Garrido-Jurado, «Generation of fiducial marker dictionaries using Mixed Integer Linear Programming,» October 2015. [En ligne]. Available: https://www.researchgate.net/publication/282426080_Generation_of_fiducial_marker_dictionaries_using_Mixed_Integer_Linear_Programming. [Accès le 26 01 2021].
- [4] «Camera Calibration and 3D Reconstruction,» OpenCV, [En ligne]. Available: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html. [Accès le 26 01 2021].
- [5] R. M. Salinas, «ArUco: An efficient library for detection of planar markers and camera pose estimation,» [En ligne]. Available: <https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITojQ76xqL7H0TEtXriJX5kwi9Kgc/edit#>. [Accès le 2021 01 26].
- [6] OpenCV, «Calibration with ArUco and ChArUco,» [En ligne]. Available: https://docs.opencv.org/master/da/d13/tutorial_aruco_calibration.html. [Accès le 27 01 2021].