

UNIVERSITÉ DE BRETAGNE OCCIDENTALE

MASTER 2 INFORMATIQUE
LOGICIELS POUR LES SYSTÈMES EMBARQUÉS

2020/2021

LAB-STICC

Coopération de drone dans un système hétérogène

Rapport de stage

Auteur :
William PENSEC

Superviseur :
M. David ESPES

Enseignante référente :
Mme Catherine DEZAN

06/04/2021 au 31/08/2021



Table des matières

| | | |
|------|--|----|
| I | Introduction | 1 |
| II | La structure d'accueil : le Lab-STICC | 1 |
| III | Présentation des objectifs du stage | 2 |
| | III.1 Contexte | 2 |
| | III.2 Problématique | 3 |
| | III.3 Fonction principale | 3 |
| | III.4 L'existant | 5 |
| | III.4.1 La plateforme | 5 |
| | III.4.2 Le drone | 7 |
| IV | Réalisation | 8 |
| IV.1 | Architectures | 8 |
| | IV.1.1 Architecture globale du projet | 8 |
| | IV.1.2 Architecture de la partie drone et explication des communications présentes | 9 |
| IV.2 | Première partie : L'automate | 11 |
| | IV.2.1 Automate : Généralités | 11 |
| | IV.2.2 Automate : Récupération des valeurs des registres | 13 |
| | IV.2.2.1 Code Java | 14 |
| | IV.2.2.2 Connexion à l'automate | 15 |
| | IV.2.2.3 Lecture des registres | 15 |
| | IV.2.2.4 Améliorations possibles | 16 |
| | IV.2.2.5 Résultat | 16 |
| | IV.2.3 Automate : Enregistrement des valeurs dans une base de données | 16 |
| | IV.2.3.1 Le Driver JDBC | 16 |
| | IV.2.3.2 Programmation et exécution de l'application | 17 |
| | IV.2.3.3 Résultat | 18 |
| | IV.2.4 Conclusion partielle | 18 |
| IV.3 | Deuxième partie : Le drone | 18 |
| | IV.3.1 Drone : Généralités | 18 |
| | IV.3.2 Drone : Équipements nécessaires | 19 |
| | IV.3.3 Drone : Onboard-SDK (OSDK) sur Raspberry Pi et tests de communications | 19 |
| | IV.3.3.1 Connexion | 19 |
| | IV.3.3.2 Résultats | 20 |

| | | |
|----------|--|----|
| IV.3.4 | Drone : Positionnement dans la pièce | 20 |
| IV.3.4.1 | Méthode de Bancroft | 22 |
| IV.3.4.2 | Méthode de Newton/Raphson | 24 |
| IV.3.4.3 | Résultats | 25 |
| IV.3.4.4 | Problèmes rencontrés | 28 |
| IV.3.5 | Drone : Déplacement dans la pièce | 28 |
| IV.3.6 | Conclusion partielle | 32 |
| IV.4 | Troisième partie : Le réseau de neurones | 32 |
| IV.4.1 | IA : Généralités | 32 |
| IV.4.2 | IA : Dataset | 33 |
| IV.4.3 | IA : Labellisation des images | 35 |
| IV.4.4 | IA : YoloV5 | 36 |
| IV.4.4.1 | Un réseau de neurones | 36 |
| IV.4.4.2 | Utilisation de Yolo | 36 |
| IV.4.4.3 | Apprentissage | 37 |
| IV.4.4.4 | Résultats | 38 |
| IV.4.5 | Conclusion partielle | 41 |
| V | Conclusion du stage | 41 |

Annexes

44

| | |
|-----------------------------------|----|
| Annexe 1 - Raspberry Pi | 44 |
| Annexe 2 - Code | 45 |

Bibliographie

51

Table des figures

| | | |
|----|---|----|
| 1 | Structure du Lab-STICC avec les différents pôles existants | 2 |
| 2 | Diagramme de Gantt représentant le temps passé sur chacune des étapes | 4 |
| 3 | Plateforme industrielle | 5 |
| 4 | Intérieur d'un automate | 6 |
| 5 | Module secondaire de la plateforme | 7 |
| 6 | Matrice 100 vue aérienne | 8 |
| 7 | Architecture Générale du projet | 9 |
| 8 | Architecture Drone/Raspberry Pi | 10 |
| 9 | Exemple d'une trame <i>UART</i> | 11 |
| 10 | Exemple de code Grafset | 12 |
| 11 | Aperçu de l' <i>IHM</i> textuelle du programme Java | 14 |
| 12 | Aperçu du contenu des registres sur le logiciel <i>Unity Pro XLS</i> | 14 |
| 13 | Capture d'écran de la base de données | 18 |
| 14 | Exemple fournis par DJI | 20 |
| 15 | Composition du réseau | 21 |
| 16 | Composition du réseau en détail | 21 |
| 17 | Placement visuel des cartes dans l'espace | 21 |
| 18 | Représentation d'une configuration avec 1 tag et 11 ancre | 24 |
| 19 | Schéma représentant le positionnement du drone et des ancre avec distance mesurée | 26 |
| 20 | Schéma représentant l'orientation du drone par rapport au point d'arrivée | 29 |
| 21 | Schéma représentant les données du capteur selon l'angle du drone . . | 31 |
| 22 | Schéma représentant l'angle à déterminer selon l'orientation du drone | 31 |
| 23 | Exemple d'image : Anomalie Bouchon | 34 |
| 24 | Exemple d'image : Anomalie Tube coloré | 34 |
| 25 | Interface graphique de labelImg | 35 |
| 26 | Exemple de labellisation avec labelImg | 36 |
| 27 | Comparaison des modèles disponible | 37 |
| 28 | Modèles disponibles pour YoloV5 | 38 |
| 29 | Matrice de confusion | 39 |
| 30 | Exemple de prédiction faite par l'apprentissage | 40 |
| 31 | Détails des pins GPIO du Raspberry Pi 3 B+ | 44 |

Liste des tableaux

| | | |
|---|--|----|
| 1 | Résultats obtenus avec Bancroft | 27 |
| 2 | Résultats obtenus avec Bancroft et Decawave | 27 |
| 3 | Résultats moyens obtenus avec Bancroft et Decawave | 28 |

Liste des blocs de code

| | | |
|----|---|----|
| 1 | Exemple d'utilisation des fonctions SQL | 17 |
| 2 | Équivalent SQL | 17 |
| 3 | Fonction de déplacement du drone | 30 |
| 4 | Exemple fonction map | 32 |
| 5 | Connexion à l'automate | 45 |
| 6 | Séparation des appels | 45 |
| 7 | Lecture des registres | 45 |
| 8 | Exemple d'exécution sur un registre de type %M | 46 |
| 9 | Exemple d'exécution sur un registre de type %MW | 47 |
| 10 | Fonction Java des requêtes SQL | 47 |
| 11 | Exemple d'exécution sur un registre de type %MW avec enregistrement dans la base de données | 48 |

Liste des abréviations

IHM Interface Homme/Machine

ISO International Organization for Standardization

OSI Open Systems Interconnection

SDK Software and Development Kit

UART Universal Asynchronous Receiver Transmitter

UWB Ultra WideBand

Remerciements

Je tiens à remercier toutes les personnes qui m'ont aidé lors de ce stage et qui ont pu m'assister dans la rédaction de ce rapport.

Tout d'abord, je tiens à remercier vivement mon maître de stage, M. David ESPES, Maître de Conférences à l'Université de Bretagne Occidentale (UBO), pour le temps passé à m'aider sur mes points de blocages. Je le remercie, par ailleurs, pour m'avoir proposé et accepté pour ce stage de fin d'études.

Je remercie également ma responsable pédagogique, Mme Catherine DEZAN, Maître de Conférence à l'UBO, pour le suivi apporté durant ce stage. De plus, je tiens à la remercier pour m'avoir aidé énormément à trouver ce stage au sein d'un laboratoire et parmi les domaines qui m'intéressent le plus.

Je remercie, par ailleurs, les quelques doctorants et quelques stagiaires du Lab-STICC qui m'ont aidé à certains moments de mon stage pour diverses raisons.

Je remercie, grandement, le corps enseignant du Département Informatique de l'Université de Bretagne Occidentale à Brest pour leur formation durant ces 3 années de Licence ainsi que ces 2 années de Master.

Enfin, je remercie toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage : M. ESPES et ma famille.

I Introduction

De nos jours, l'industrie devient de plus en plus automatisée. Il est donc nécessaire d'apporter un suivi efficace à ces automatisations et aux chaînes de productions automatisées afin d'intervenir et de prévenir de potentielles pannes sur le système qui peuvent occasionner une cessation de production selon la gravité.

C'est pourquoi une solution envisageable à long terme est la prévention de pannes par drones. Leurs avantages c'est qu'ils sont très mobiles, petits et très efficace dans des milieux exigus. Ils sont facile d'accès et peuvent effectuer de très nombreuses tâches variées.

Nous assistons depuis quelques années au fort développement et à l'accessibilité croissante des drones sur le marché public. Ils étaient jusque là réservé à de très petites quantités de personnes telles que les militaires ou les laboratoires de recherche. Aujourd'hui, il est de plus en plus courant d'avoir chez soi un drone personnel.

C'est pourquoi, dans le cadre de mon Master 2 LSE (Logiciels pour les Systèmes Embarqués) et également avec ma volonté de continuer sur les métiers de la Recherche, je me suis orienté vers un stage en Laboratoire sur le domaine des drones, automatisations de tâches, prévention de risques avec détection, analyse et traitement.

II La structure d'accueil : le Lab-STICC

Le Lab-STICC^[1], CNRS UMR 6285, est le Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance. Il est présent sur les villes de Brest, Quimper, Lorient et Vannes. Il est rattaché à l'Institut des Sciences de l'Information et de leurs Interactions (INS2I) et à l'Institut des Sciences de l'Ingénierie et des Systèmes (INSIS) du CNRS. Les pôles du laboratoire sont définis comme des entités permettant d'établir une cohérence scientifique sur les grands domaines couverts par le Lab-STICC tels que par exemple : l'IA et l'Océan, les Systèmes Photoniques et les Hyperfréquences, CyberSécurité et les Réseaux ou encore les Interactions entre les Systèmes Humains - Machines . Il existe 9 pôles différents au sein du laboratoire.

J'étais assigné au sein du pôle CyR (Cyber et Réseaux) et de l'équipe IRIS (Sécurité et Résilience des Systèmes d'Information).

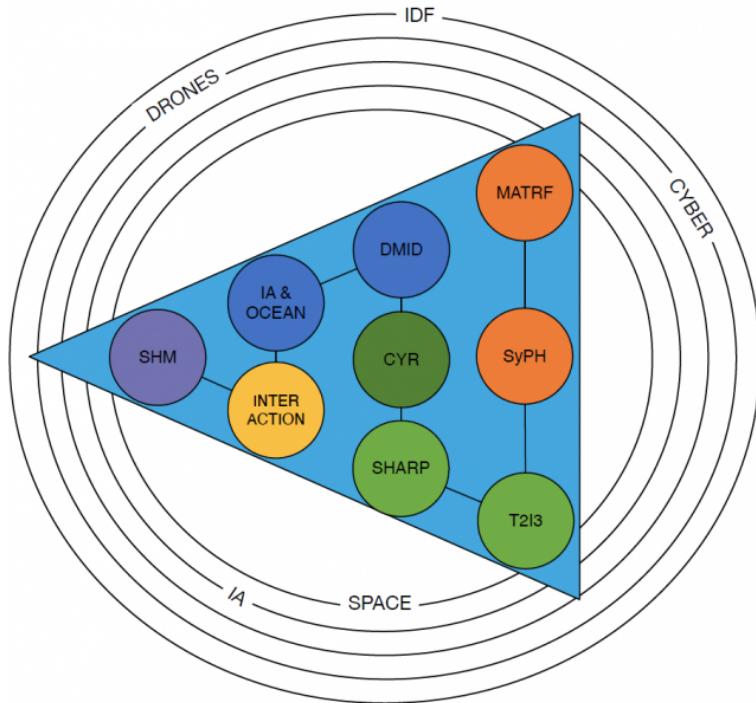


FIGURE 1 – Structure du Lab-STICC avec les différents pôles existants

III Présentation des objectifs du stage

III.1 Contexte

De nos jours, dans le contexte de l'industrialisation et de la numérisation des productions, il est important d'avoir des réponses automatisées afin de gagner en flexibilité et en compétitivité. Une automatisation de ce type repose donc sur des systèmes et des équipements très hétérogènes que ce soit dans les systèmes employés, les informations à analyser ou encore les protocoles réseaux utilisés.

Ces infrastructures reposent sur des systèmes cyber-physiques. Bien que ces infrastructures assurent une traçabilité très précise de la chaîne de production, les informations récoltées peuvent ne pas suffire afin d'identifier un potentiel problème et son origine.

Les domaines d'applications des drones peuvent être très divers et peuvent être complémentaires à d'autres systèmes de reconnaissance déjà présent. Les drones peuvent par exemple être utilisés dans la surveillance de l'infrastructure ou encore compléter des diagnostics qui peuvent être réalisés à distance par des opérateurs en cas de panne. Les drones étant équipés de caméra et pouvant se déplacer librement sans interférer avec le processus de fabrication, ils sont des atouts indispensables pour prendre des images de la panne et les envoyer aux opérateurs pour le traitement, l'analyse et le recouvrement de la panne.

Lorsque le drone se déplace pour aller jusqu'à la panne, il doit être en mesure de se localiser précisément afin de s'y rendre le plus rapidement possible tout en évitant de causer des dommages corporels avec du personnel se trouvant sur le chemin. Pour se faire, le drone doit être synchronisé avec la chaîne de production et utiliser des algorithmes de détection de panne qui seraient implantés directement dans le système afin de l'avertir d'une panne immédiate ou d'une potentielle menace de panne si des algorithmes de maintenances sont utilisés.

III.2 Problématique

La problématique de ce stage est d'assurer une coopération entre les dispositifs présents tels que la chaîne de production, les automates et le drone ainsi que d'assurer différentes opérations comme la prise d'image d'une panne au sein de la plateforme ou encore la surveillance de l'environnement afin d'éviter des risques pour le personnel ou les équipements.

III.3 Fonction principale

Ce stage s'est composé de trois grandes étapes principales.

La première étape était la découverte de l'environnement. Cette étape a consisté à comprendre le fonctionnement des automates et de la plateforme. La compréhension de l'environnement s'est faite grâce aux protocoles réseaux, au système, et les langages de programmation utilisés. J'ai ainsi découvert le fonctionnement d'un automate programmable ainsi que le protocole de communication ModBus TCP/IP.

La seconde étape a été la connexion du drone à la partie industrielle (la plateforme). Le laboratoire Lab-STICC possède un drone de la marque DJI : le drone Matrice 100 (M100)^[2]. Il m'a fallu analyser les capteurs du drone ainsi que trouver une solution afin de le piloter dans l'espace contraint de la pièce représentant l'industrie. Le drone pouvait être équipé de cartes de communication sans fil afin d'interagir avec la plateforme, d'une caméra afin de prendre des photos des dispositifs, ou de tout autre équipement nécessaire au bon fonctionnement du drone dans la pièce par rapport à nos besoins.

La dernière étape était l'analyse de données par le drone. Le but était ainsi de créer une interaction entre le drone et la plateforme afin que le drone se déplace d'un point initial vers un point où se trouve une panne qui a été détectée ou estimée, puis prendre des photos de la panne et les envoyer aux opérateurs. Le chemin que prend le drone doit être calculé afin qu'il ne comporte aucun risque pour l'environnement (personnels ou équipements).

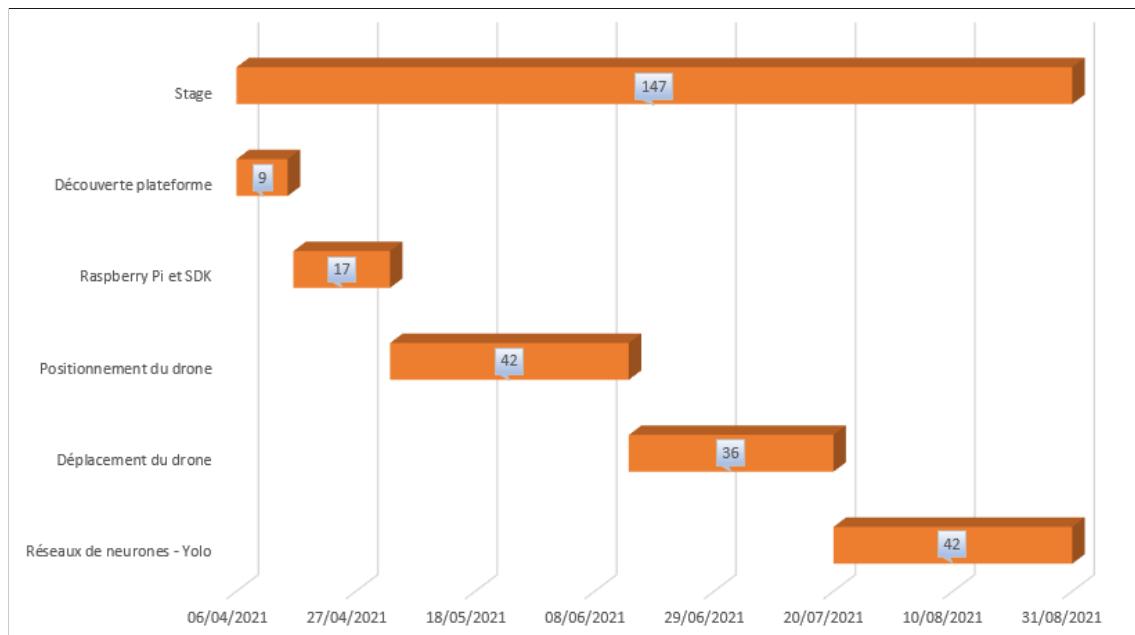


FIGURE 2 – Diagramme de Gantt représentant le temps passé sur chacune des étapes

III.4 L'existant

III.4.1 La plateforme

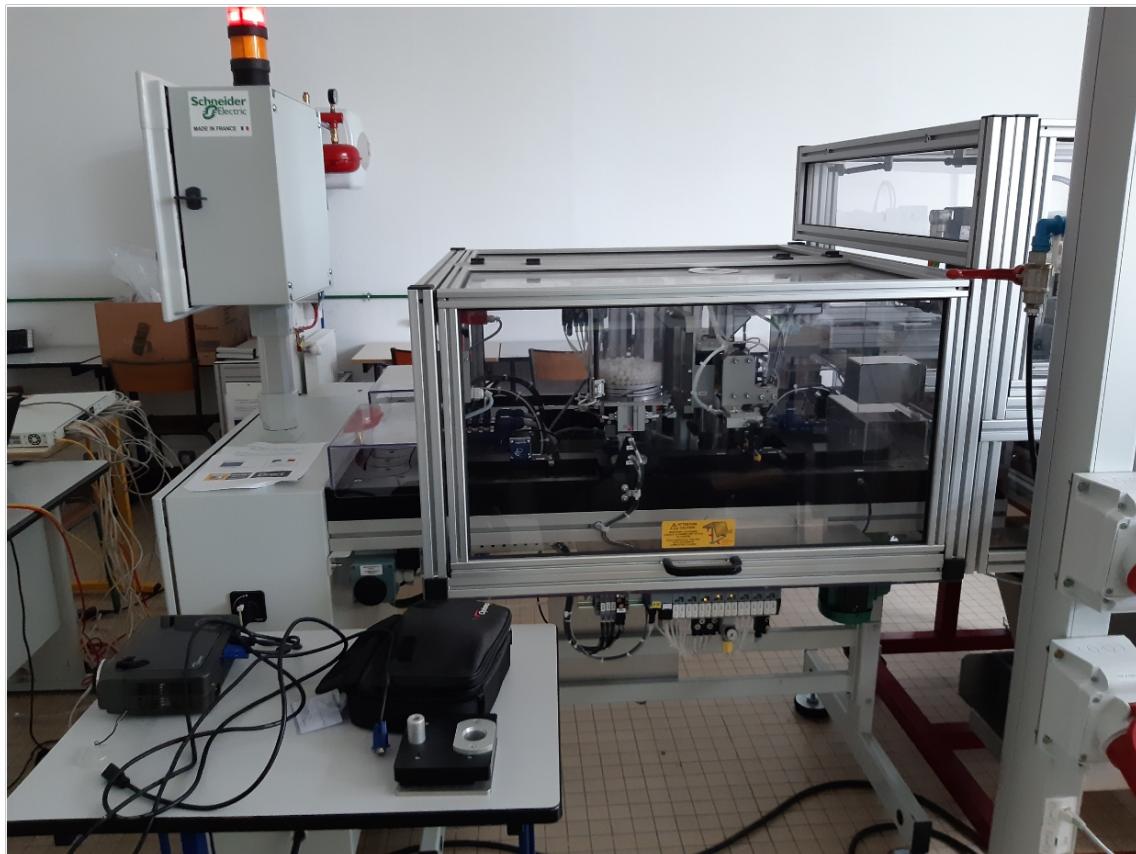


FIGURE 3 – Plateforme industrielle

Nous avons à disposition au laboratoire une réplique de plateforme (voir figure 3) d'un complexe d'industrie pharmaceutique. Cette plateforme est conçue initialement par Schneider Electric. Elle se décompose en deux modules comportant chacun un automate de la même marque (automate M340 et M580). Un exemple de l'intérieur de l'automate se trouve sur la figure 4.

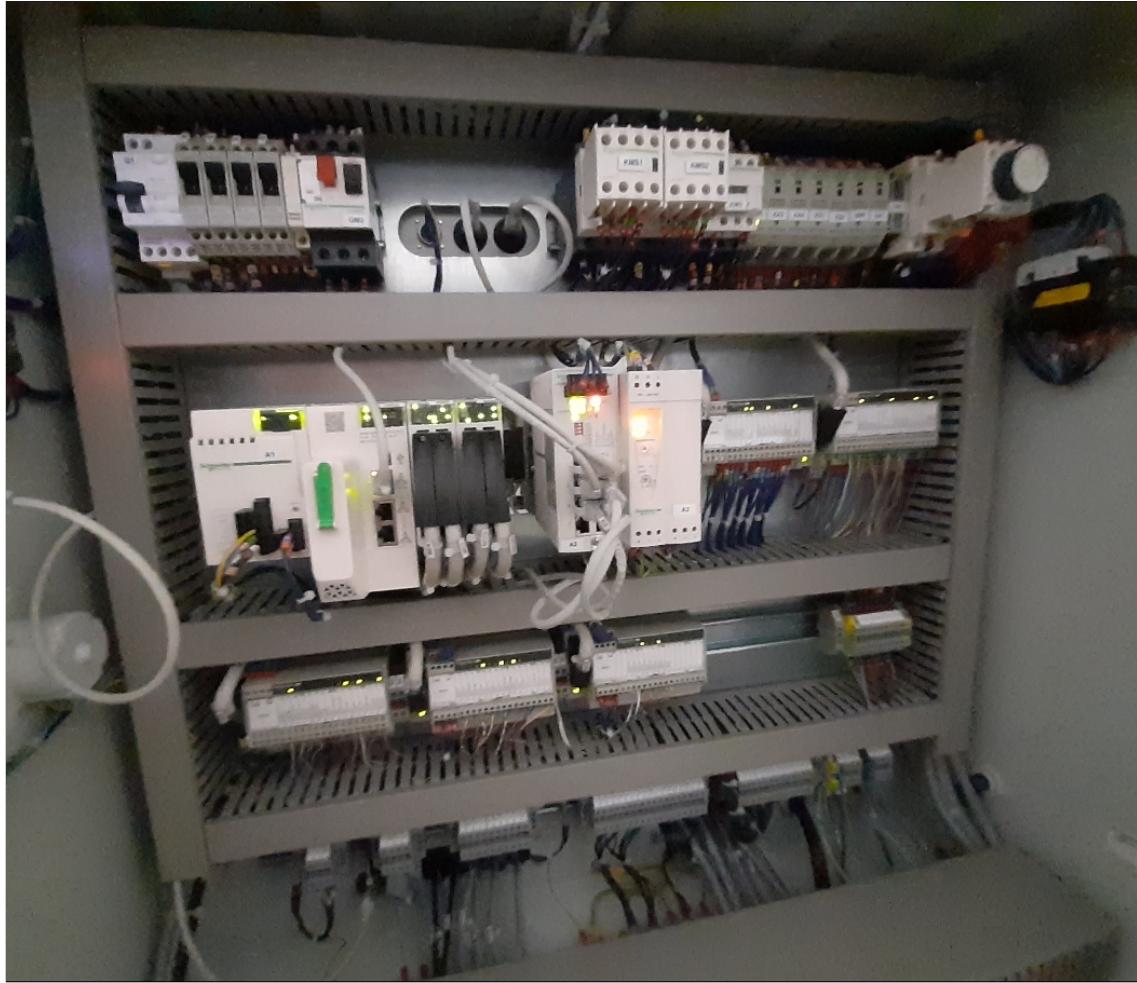


FIGURE 4 – Intérieur d'un automate

Le premier module s'occupe de la gestion des commandes (recettes) dictées par l'utilisateur. Une fois une commande réalisée, un bras articulé va venir récupérer un flacon et un bouchon puis va les déposer sur une palette. Ensuite, cette palette va passer sur un convoyeur et aller dans le deuxième module. Lorsque la palette revient du second module, le bras articulé récupère le flacon bouché et met ce flacon dans un carton. Une fois que le carton est plein, un convoyeur amène le carton dans un autre endroit afin d'y être stocké.

Le second module s'occupe du traitement de la commande. La commande est traitée par des puces RFID qui se trouvent tout au long de la chaîne de production. La commande consiste à mettre des billes dans le flacon. La quantité de billes à mettre dépend de la commande. Comme on peut voir sur la figure 5, la partie gauche avec le réservoir de bille et à droite l'appareil qui va refermer le flacon avec le bouchon mis sur la palette.



FIGURE 5 – Module secondaire de la plateforme

III.4.2 Le drone

Le drone possédé par le laboratoire est le matrice M100 de la marque DJI. Ce drone est optimal pour le monde de la recherche car il possède de nombreux capteurs, un bon autopilote, de nombreux tutoriels et également la possibilité d'embarquer davantage de matériels comme une caméra, des batteries, un ordinateur embarqué, etc. L'entreprise DJI met à disposition des outils afin d'accéder aux capteurs, autopilote et autres fonctionnalités de leurs drones. Ces outils peuvent être appliqués pour créer une application Android/iOS pour manipuler le drone, gérer les capteurs/appareils ajoutés au drone, ou encore de connecter son propre ordinateur embarqué à un drone compatible par le port série (*UART*).

Selon le besoin de ce stage, j'ai utilisé l'outil fourni par DJI s'appelant Onboard-SDK[3]. Cet outil permet de connecter un ordinateur embarqué sur le drone et d'accéder aux capteurs, à l'autopilote et de manipuler le drone à distance de manière automatique et programmable directement dans l'ordinateur en C++. Il existe de nombreux exemples sur le site du constructeur afin de faire fonctionner ce *SDK* sur le drone de notre choix. Le choix de la version du *SDK* est déterminé par le drone. Nous utilisons un drone Matrice 100 dont la dernière version du *SDK* compatible est la version 3.9 du *SDK*. Les versions suivantes peuvent être compatibles également mais elles ne sont pas indiquées comme tel sur le site du constructeur et donc afin d'anticiper un quelconque problème de compatibilité j'ai préféré prendre la version

recommandée par le constructeur.



FIGURE 6 – Matrice 100 vue aérienne

IV Réalisation

IV.1 Architectures

Dans un premier temps, je parlerai de l'architecture globale du projet. Puis dans une seconde partie, je parlerai plus précisément de l'architecture du point de vue du drone en terme de communications et de protocoles utilisés.

IV.1.1 Architecture globale du projet

La figure 7 représente l'architecture générale du projet avec les différents composants utilisés, les connexions entre les composants et les protocoles de communications utilisés. Cette figure représente le projet d'un point de vue très général. J'aborderai dans la partie IV.1.2 la partie spécifique au drone et sa communication avec les autres composants du projet.

L'ordinateur Client communique avec la plateforme via le protocole de communication ModBus. L'application Java est exécutée sur le client et accède en lecture et écriture à la base de donnée intégrée également au client. Le drone accède à la base de données en WiFi et se positionne grâce à des cartes Decawaves utilisant le protocole *UWB*. Le Raspberry Pi connecté sur l'*UART* du drone permet la communication entre le *SDK* du drone et le système entier, ce qui permet de faire gérer le déplacement du drone de manière automatisée entre un point de départ et un point d'arrivée qui peut correspondre à un capteur défectueux sur la plateforme ou sur lequel une anomalie a été détectée.

Le protocole *UWB* se définit par une technologie radio pouvant utiliser un très faible niveau d'énergie. Le rayon d'action est de courte portée. Les applications traditionnelles de l'*UWB* sont l'image radar, la collecte de données de capteurs, la localisation et des applications de suivi. La prise en charge de l'*UWB* a commencé à apparaître très récemment dans les Smartphones haut de gamme comme l'iPhone 11 ou le Samsung Galaxy Note20 Ultra, et les AirTag d'Apple. La bande de fréquence d'utilisation commence vers 3.1GHz et va jusqu'à 10.6GHz et ne gêne théoriquement pas les autres communications situées dans ces même bandes.

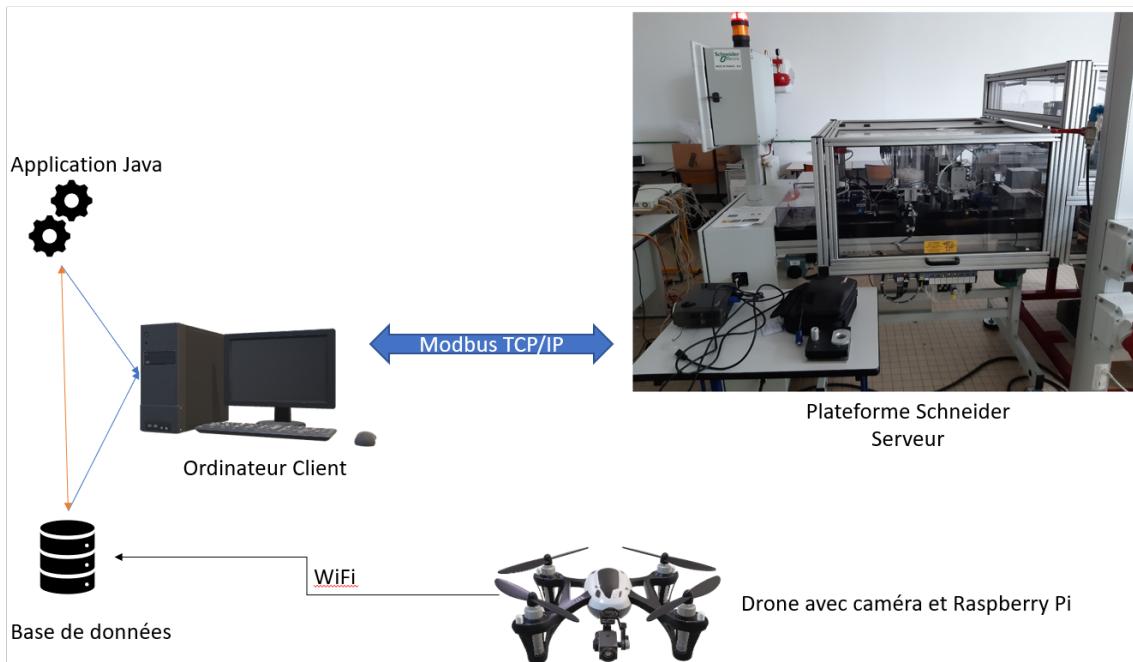


FIGURE 7 – Architecture Générale du projet

IV.1.2 Architecture de la partie drone et explication des communications présentes

A présent, la figure 8 correspond à la partie du drone avec le Raspberry Pi 3 B+ lié au drone via une connexion *UART* (pin 8 et 10 (RX/TX) du Raspberry Pi : voir figure 31 pour plus de détails). Par ailleurs, une carte Decawave est connectée sur

le Raspberry Pi afin de pouvoir se positionner dans la pièce en communiquant avec 4 autres cartes placées dans la pièce en utilisant une communication *UWB*.

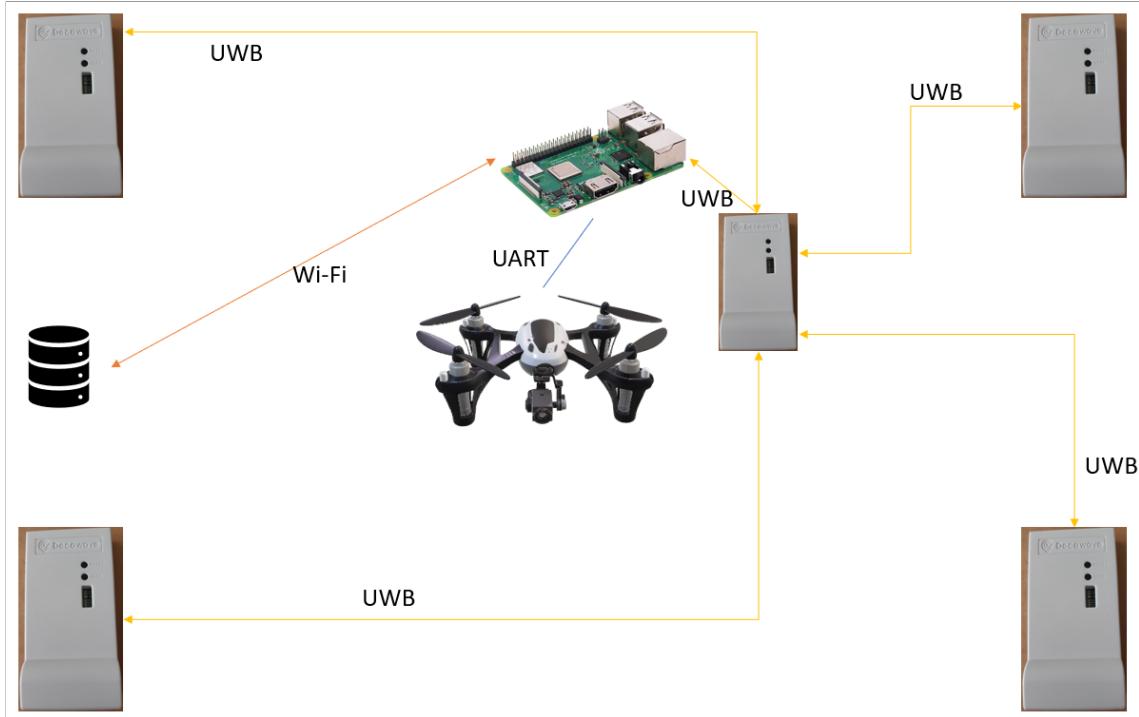


FIGURE 8 – Architecture Drone/Raspberry Pi

La connexion *UART*[4] correspond à une connexion entre le Raspberry Pi et le port série du drone. La vitesse de transmission de ce port série est de 230400 bauds dans mon cas afin d'utiliser le Raspberry Pi et le *SDK* du drone.

Un trame *UART* est composé de deux paramètres.

Le premier paramètre est un chiffre correspondant au nombre de bauds[5]. C'est à dire la vitesse de transmission, un baud représente la fréquence de (dé)modulation d'un signal, c'est-à-dire le nombre de fois où ce signal change par seconde. Il peut être égal au nombre de bit transmis par seconde si l'on considère à tort que la vitesse de transmission et le taux de transfert équivalent à la même chose.

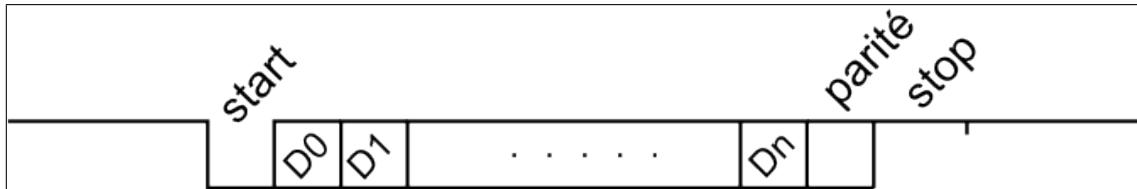
Le second paramètre est composé de trois sous-paramètres. Ils se décomposent en un chiffre, une lettre (I, P, N) et un autre chiffre. Le premier chiffre correspond au nombre de bits de données. La lettre correspond au bit de parité :

P : Parité paire

I : Parité impaire

N : Pas de parité

Enfin, le troisième sous-paramètre correspond à la durée du bit stop. La durée varie entre 1, 1.5 ou 2.

FIGURE 9 – Exemple d'une trame *UART*

Voici quelques exemples de paramétrage d'*UART* :

- 4800 4P1 (EN "4800 4E1") : 4800 bauds, 4 bits de données, parité paire (P)(E comme "even" en anglais), 1 bit de stop.
- 9600 7I2 (EN "9600 7O2") : 9600 bauds, 7 bits de données, parité impaire (I)(O comme "odd"), 2 bits de stop.
- 115200 8N1 : 115200 bauds, 8 bits de données, pas de parité (N), 1 bit de stop.

Le Raspberry Pi communique avec les cartes Decawaves afin de se positionner dans la pièce. Les cartes qui sont utilisées dans ce stage sont les cartes *MDEK1001*[6]. Le protocole de communication utilisé par ces cartes est l'Ultra Large Bande ou Ultra WideBand (*UWB*). Une carte Decawave est relié au Raspberry Pi via USB. Ils communiquent par le port USB en liaison série avec une vitesse de transmission de 115200 bauds. Il y a 4 autres cartes connectées et actives dans la pièce. Il existe deux types de cartes, les ancrés qui correspondent à des cartes immobiles dont on connaît la position car elle sera fixée par l'utilisateur grâce à l'application Android/iOS fournie par Decawave[7] et les tags qui correspondent aux cartes mobiles.

IV.2 Première partie : L'automate

IV.2.1 Automate : Généralités

L'automate sur lequel j'ai travaillé était un automate venant de chez Schneider Electric comme abordé dans la partie III.4.1. Il est accessible en se connectant via le réseau local et on communique avec l'automate grâce au protocole de communication ModBus. Ce protocole a été créé par l'entreprise Modicon en 1979. Cette société a ensuite été achetée en 1996 par Schneider Electric. Il est utilisé pour les automates programmables et s'applique sur la couche applicative (Niveau 7 du modèle *OSI*¹). Ce protocole est basé sur une structure hiérarchisée comme par exemple : Maitre/Escalves ou Client/Serveurs. Pour mon stage, j'utilisais ce protocole en mode TCP/IP et donc Client/Serveur. Le client était déterminé par l'ordinateur sur lequel je travaillais et c'est via cet ordinateur que j'accédais à l'automate via des lectures et écritures dans le serveur qui était l'automate. Le serveur est identifié par une adresse

1. Le modèle *OSI* est une norme de communication de tous les systèmes informatiques proposé par l'*ISO*.

IP et un numéro de port sur lequel il attend les demandes de connexion (port 502 par défaut). Le port du client sera lui toujours supérieur à 1024 afin de ne pas se superposer avec les ports définis pour le système et sera différent pour chaque client connecté simultanément.

Pour le programmer, récupérer les variables, les registres ou modifier le fonctionnement, il est nécessaire d'utiliser le logiciel UNITY PRO XLS et de programmer l'automate en Grafcet (GRAphe Fonctionnel de Commande Étapes / Transitions) ou en C. Ce langage de programmation est un mode de représentation d'un automatisme très bien adapté aux systèmes séquentiels. Il est possible de comparer le Grafcet au modèle mathématique des réseaux de Petri. Le Grafcet est un langage graphique qui représente le fonctionnement d'un automate par une suite d'étapes associées à des actions et des transitions entre chaque étape (voir l'exemple à la figure 10).

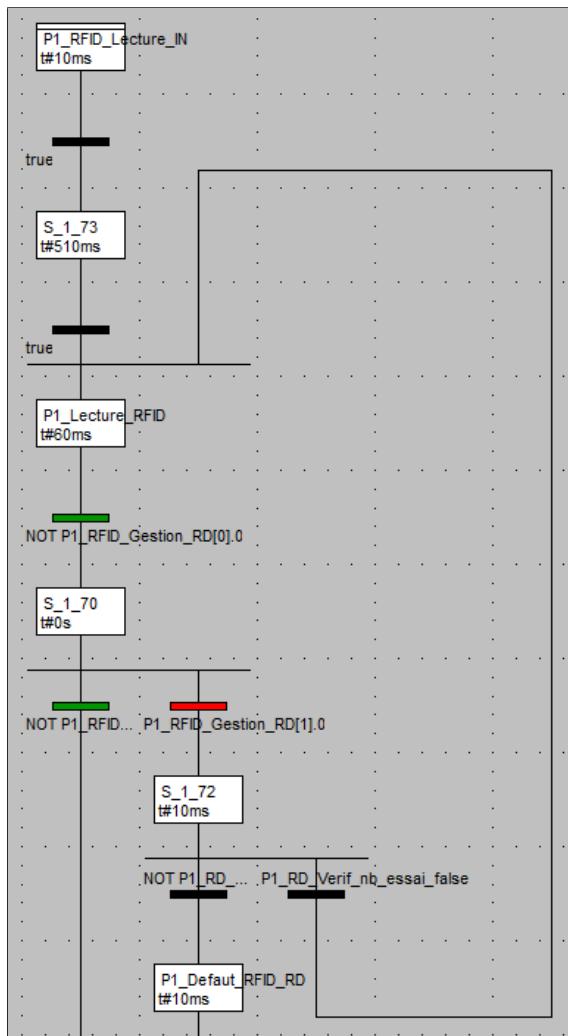


FIGURE 10 – Exemple de code Grafcet

Les automates de la plateforme sont composés de registres. Chaque registre est déterminé par son adresse. Il y a plusieurs types de registre :

- %I : représente une entrée (Input) ; elle est mise à jour dans la mémoire en début de tâche ou quand l'automate est en mode RUN ou STOP. Exemple : $\%I0.2.39.0$ correspond à la valeur d'entrée pour la présence du flacon sur la palette.
- %Q : représente une sortie ; elle est mise à jour à la fin de la tâche, uniquement lorsque l'automate est en mode RUN. Exemple : $\%Q0.3.16.0$
- %M : représente un mot de taille 1 (type bit mémoire). Exemple : $\%M103$ correspond à la présence d'une palette ou non sous le récipient à bille n°1.
- %MW : représente un mot d'une taille N (type mot mémoire). Exemple : $\%MW150$ correspond à la lecture de la puce RFID au niveau du poste 1 (récipient à bille).

Il existe d'autres types de registres mais je ne les ai pas utilisés, voir ils pouvaient ne pas être utilisé du tout dans la base de registre à ma disposition.

Dans mon cas, j'ai dû lire les valeurs des registres seulement dans les registres de type %M et %MW.

IV.2.2 Automate : Récupération des valeurs des registres

J'ai créé un programme Java me permettant, via une interface homme/machine (voir figure 11), de récupérer les valeurs des registres de l'automate depuis l'ordinateur qui sert de Client. J'ai inscrit des valeurs par défaut dans le code afin de simplifier et accélérer l'exécution du programme lorsque l'on désire faire plusieurs relevés avec peu de délai entre chaque exécution.

Les valeurs que je récupère depuis les registres de l'automate sont également accessibles via le logiciel *Unity Pro XLS*, comme dit dans la partie IV.2.1. Comme on peut voir à la figure 12, le registre s'appelant 'P1_Presence_palette_img' correspond au registre $\%M103$ qui est de taille 1 ; quant au deuxième registre se nommant 'P1_RFID_Gestion_RD', il correspond au tableau de registre de taille 4 $\%MW150$ et c'est seulement la première valeur qui change à chaque fois que la palette fait un tour sur le convoyeur visible à la figure 5.

```

Adresse de l'automate (défaut : 192.168.1.4) :

Temps entre chaque lecture (défaut : 5000ms) :
1000
Nombre de répétitions de la lecture (défaut : 1) :
5
Adresse du registre à lire :
%MW150
Taille du registre à lire (défaut : 1) :
5
Données enregistrées :
192.168.1.4 | 1000 | 5 | 150 | 5
Si c'est OK, entrez 'ok'
ok
-----
```

FIGURE 11 – Aperçu de l'IHM textuelle du programme Java

| Nom | Valeur | Type |
|--------------------------|--------|-------------------|
| P1_Presence_palettes_img | 0 | EBOOL |
| P1_RFID_Gestion_RD | | ARRAY[0..3] OF... |
| P1_RFID_Gestion_R... | 12800 | INT |
| P1_RFID_Gestion_R... | 0 | INT |
| P1_RFID_Gestion_R... | 10 | INT |
| P1_RFID_Gestion_R... | 38 | INT |

FIGURE 12 – Aperçu du contenu des registres sur le logiciel *Unity Pro XLS*

IV.2.2.1 Code Java

Afin de programmer l'application, il me fallait me connecter à l'automate, renseigner l'adresse du registre, le type de registre et la taille du registre à lire, le nombre de fois que l'on veut lire ce registre et à quel intervalle.

Les valeurs par défaut sont inscrites dans le code et s'affichent lors de l'exécution. Pour modifier une valeur par défaut il suffit de rentrer une valeur au clavier, elle apparaîtra en vert (voir image 11) dans la console.

Une fois les valeurs rentrées lors de l'exécution, un message de confirmation apparaît afin de bien vérifier les valeurs et qui permet de corriger une valeur s'il y a

une erreur au lieu de tout relancer. Une fois la confirmation faite, le programme se lance.

Tout d'abord, l'application lance la connexion sur l'adresse de l'automate donnée dans les valeurs précédentes et sur le port 502 qui est celui par défaut (voir le détail dans la partie IV.2.2.2). Si la connexion est réussie, il affichera *Client connecté* sinon il affichera un message d'erreur correspondant au type d'erreur rencontrée comme par exemple un client déjà connecté à l'automate, une connexion impossible (câble ethernet débranché par exemple).

Ensuite, si la connexion avec l'automate est réalisée, la lecture de la variable peut se faire (voir la partie IV.2.2.3 pour plus de détails).

Enfin, il y a déconnexion entre le serveur et le client et affichage du message de retour : *Client déconnecté* ou *Client non déconnecté*.

Dans les prochaines pages, des morceaux de code seront présentés. Le code est mis tel que je l'ai dans mon programme. Des parties peuvent être abordées sans expliquer totalement le code mis ; mais les parties non expliquées le seront dans les paragraphes suivants.

IV.2.2.2 Connexion à l'automate

Pour connecter le client au serveur, il fallait établir un protocole de communication ModBus. J'ai ainsi, dans mon programme Java, incorporé une librairie qui gère le protocole ModBus afin de communiquer avec l'automate. Pour se faire, j'ai utilisé la librairie *j2mod*[8].

Cette librairie est très simple d'utilisation et possède un tutoriel sur GitHub afin de démarrer et commencer à communiquer[9].

Afin de lancer la connexion avec le serveur, il suffit d'appeler le constructeur puis une fonction comme on peut voir à la ligne 4 et 5 du bloc de code 5 en annexe 2. Le constructeur prend en paramètre l'adresse IP du serveur. Dans mon cas, j'ai stocké cette adresse dans un attribut de la classe et donc je passe seulement cet attribut au constructeur. La fonction générale que l'on peut voir sur le bloc 5 en annexe 2 renvoie, si la connexion s'est bien exécutée ou non afin d'afficher à l'utilisateur, un message visuel dans la console.

IV.2.2.3 Lecture des registres

Une fois que la connexion est faite entre le Client et le Serveur, il est possible de récupérer la valeur d'un registre. Dans mon code j'ai créé une fonction principale qui va lire un registre de taille N . La petite difficulté est qu'il y a deux types de registres différents à lire et de tailles différentes. Un des registres sera toujours de taille 1 et l'autre est un tableau de N cases. Ainsi, la fonction principale, appelée

`readVar()`, permet la séparation des appels selon le type de registre et permet également la répétition de la lecture selon le nombre de répétitions et le délai demandé par l'utilisateur.

Une fois que le type de registre est connu du programme, il peut appeler la fonction correspondante afin de lire le contenu selon la taille. La taille du registre influera sur la fonction de la librairie. Une fois l'appel de la fonction fait, le résultat est affiché dans la console pour l'utilisateur.

IV.2.2.4 Améliorations possibles

Les améliorations possibles de cette partie sont de formater le code professionnellement. Par exemple, en gérant les exceptions dans une classe séparée du code et non pas directement dans le code comme actuellement.

IV.2.2.5 Résultat

Comme il est possible de voir sur les blocs 8 et 9 en Annexe 2, l'exécution du programme se fait via une interface minimale textuelle et affiche les résultats à la ligne à chaque lecture.

IV.2.3 Automate : Enregistrement des valeurs dans une base de données

Dans cette nouvelle sous-partie, je vais évoquer les améliorations du programme que j'ai abordé dans la partie IV.2.2.1. Le but principal du projet est, rappelons le, la surveillance d'une plateforme de manière automatisée avec traitement et analyse des erreurs survenues.

Ainsi, pour se faire, il est nécessaire de vérifier en continu les valeurs des registres comme nous l'avons abordé dans la partie IV.2.2. Puis ensuite, de stocker ces valeurs récupérées dans une base de données qui sera accessible au drone afin d'analyser ces valeurs et détecter une potentielle erreur qui serait survenue avant d'intervenir (voir la partie IV.3).

C'est pourquoi, j'ai itéré une première fois la version du programme Java de la partie IV.2.2 en y ajoutant cette fois une partie de stockage et lecture de la base de données. La base de données que nous utilisons est une base de données en local directement installée sur l'ordinateur "Client", elle est de type 'Microsoft SQL Server 2014' et on y accède via *localhost* sur le port 49190.

IV.2.3.1 Le Driver JDBC

Pour travailler avec une base de données en Java, il est nécessaire d'utiliser un driver qui permettra de faire le lien entre la base de données et l'application Java. Le driver est dépendant du type de base de données. Étant donné que j'utilise une base de données de type 'Microsoft SQL Server 2014', je suis allé récupérer le driver sur le site de Microsoft[10].

IV.2.3.2 Programmation et exécution de l'application

Une fois le driver installé selon la version Java du programme (version 8 de Java pour ma part), il faut indiquer dans les attributs de connexion l'URL avec une forme précise : `jdbc:sqlserver://localhost//SQL2014:49190;`. Ensuite dans la même chaîne de caractère, on indique le nom de la base (`sensorData` pour moi), l'utilisateur et le mot de passe. Cette chaîne de caractère donne donc pour moi : `jdbc:sqlserver://localhost//SQL2014:49190; database=sensorData; user=XXX; password=XXX;`.

Le programme se connecte à la base de données via cette chaîne de caractère puis ensuite, il est possible de lancer des requêtes SQL tel que des *INSERT*, *SELECT* ou *DELETE*. J'ai créé des fonctions permettant de gérer les requêtes SQL de type *INSERT* et *SELECT*.

La fonction d'insertion (voir bloc de code n°10 en Annexe 2) prend en paramètre une chaîne de caractère et un objet. La chaîne de caractère servira à déterminer le nom du registre à insérer et l'objet est une valeur numérique qui correspondra à la valeur du registre au moment de la lecture.

La fonction de sélection prend en paramètre également une chaîne de caractère correspondant à ce que l'on veut récupérer de la base de données (tout, une colonne, ...) et une chaîne de caractère correspondant aux options possibles dans une requête SQL. Les options peuvent être par exemple *WHERE*, *ORDER BY*, *GROUP BY*, etc. Ces fonctions sont appelées dans les fonctions de lecture des variables comme on peut voir sur le bloc de code 6 en Annexe 2 pour la fonction *select()* ou sinon dans les sous-fonctions de lecture spécifique (voir le bloc de code 7 en Annexe 2) pour la fonction *insert()*.

Un exemple d'appel de ces fonctions peut être :

```

1 requete.insert("MW150", 1);
2 requete.select("*", "WHERE registerName = 'MW150'");
```

Listing 1 – Exemple d'utilisation des fonctions SQL

Ce qui équivaut à faire en SQL :

```

1 INSERT sensorValue (registerName, registerValue) VALUES ('MW150', '1');
2 SELECT * FROM sensorValue WHERE registerName = 'MW150';
```

Listing 2 – Équivalent SQL

IV.2.3.3 Résultat

Comme il est possible de voir sur le bloc 11 en Annexe 2 et la figure 13, l'exécution du programme et l'enregistrement dans la base de données se fait bien. Il est possible de lire les données des registres de l'automate et de les sauvegarder dans la base de données.

| | currDateTime | registerName | registerValue |
|---|-------------------------|--------------|---------------|
| 1 | 2021-08-19 15:09:49.700 | MW150 | 6400 |
| 2 | 2021-08-19 15:09:59.920 | MW150 | 12800 |
| 3 | 2021-08-19 15:10:10.147 | MW150 | 19200 |
| 4 | 2021-08-19 15:21:56.080 | M103 | 0 |
| 5 | 2021-08-19 15:21:58.280 | M103 | 1 |

FIGURE 13 – Capture d'écran de la base de données

IV.2.4 Conclusion partielle

L'objectif de cette partie était de se connecter à l'automate, lire les valeurs des registres puis sauvegarder cette valeur dans une base de données afin que le drone, que l'on abordera dans la partie suivante, puisse accéder à cette valeur et détecter s'il y a une anomalie qui s'est produite ou non.

Pour se faire, j'ai ainsi créé un programme Java qui se connecte à l'automate selon des paramètres donnés par l'utilisateur. Dans un second temps, il va lire la valeur du registre ou les valeurs si le registre demandé est un tableau. Enfin pour terminer, le programme sauvegarde la valeur du registre lu dans une base de données de type 'Microsoft SQL Server 2014' afin que la valeur soit accessible plus tard par le drone ou par un utilisateur de la chaîne de production.

IV.3 Deuxième partie : Le drone

IV.3.1 Drone : Généralités

Comme dit dans la partie III.4.2, le laboratoire possède 2 drones achetés récemment. Les deux drones sont identiques et ce sont les DJI Matrice 100[11]. Ces drones sont indiqués par le constructeur comme *fait pour les développeurs*. Il y a ainsi de nombreuses applications possibles grâce à ces drones. La communication avec le drone se fait de diverses manières, selon ce que l'on veut faire. Par exemple, pour une utilisation classique d'un drone, il y a la télécommande de livrée et il est possible

de diriger le drone grâce à cette télécommande ; ou sinon, si on veut utiliser l'auto-pilote, il est possible de donner, grâce à l'application Android DJI Go, des points GPS au drone et il exécutera la course en suivant les points GPS donnés puis en retournant au point de départ. Ou enfin, une autre solution, celle que j'ai appliquée, c'est d'utiliser un ordinateur embarqué tel qu'un Raspberry Pi ou le Manifold[12] qui est construit et développé par la même entreprise, DJI. L'avantage du Raspberry Pi sur n'importe quel autre ordinateur embarqué c'est la puissance disponible par rapport au prix. J'ai eu à disposition un Raspberry Pi 3B+ (RPi3) qui comporte 1Go de RAM, processeur Cortex-A53 cadencé à 1.4GHz en 64 bits, avec tout un large éventail de connexions réseaux, une possibilité d'avoir une sortie vidéo et sonore. Aujourd'hui, un RPi3 coûte environ 68€ (avec carte SD, alimentation) ce qui est négligeable par rapport aux possibilités qu'il offre et surtout par rapport au rapport qualité/prix sur tous ses concurrents.

IV.3.2 Drone : Équipements nécessaires

Afin d'effectuer les missions nécessaires au stage, il est utile d'utiliser un ordinateur embarqué tel que le Raspberry Pi cité précédemment. Une fois ce mini-ordinateur choisi, on peut imaginer le matériel nécessaire, par exemple le positionnement du drone dans la pièce. Nous avons un étage au dessus du plafond de la salle et énormément de matériel métallique dans la pièce utilisée, ce qui conduit à une impossibilité d'utiliser le GPS ainsi que la centrale inertuelle (voir partie IV.3.4). De ce fait, nous avons utilisé des cartes Decawave fonctionnant en *UWB* ce qui permet de positionner le drone rapidement en sachant que 12 cartes plus l'application afin d'utiliser ces cartes coutent 199\$. Enfin, nous avons besoin d'une caméra afin de prendre en photo les différents capteurs si l'on détecte une anomalie. Pour la caméra, le choix a été un peu long car il y a beaucoup de choix : GoPro, caméra DJI, caméra Raspberry Pi. Chacune de ces caméras avait leurs avantages et leurs inconvénients. La caméra DJI coûte très cher (entre 1000€ et 3000€) mais donne une image très nette car la caméra est stabilisée ; la GoPro coûte bien moins cher mais reste onéreuse (330€ avec les fonctionnalités voulues) ; et la caméra du Raspberry Pi coûte bien moins cher encore mais n'est pas stabilisée donc des photos moins nette mais pour environ 57€. Au final, la GoPro était impossible à utiliser car la GoPro nécessitait une connexion WiFi pour communiquer avec le Raspberry Pi mais la communication WiFi était déjà prise sur la carte pour pouvoir accéder à la base de données sur l'ordinateur. Nous nous sommes donc orienté vers la caméra du Raspberry Pi car les fonctionnalités qu'elle possède nous suffisait et il est simple de capturer des images depuis un programme en C++.

IV.3.3 Drone : Onboard-SDK (OSDK) sur Raspberry Pi et tests de communications

IV.3.3.1 Connexion

L'outil de communication avec le drone fourni par DJI se nomme Onboard-SDK (OSDK). Il est utilisé dans un ordinateur embarqué (Raspberry Pi) connecté au drone.

Tout d'abord, avant de pouvoir utiliser le OSDK sur le Raspberry Pi, il faut régler certains paramètres tels que le *Baud Rate*, *la fréquence de transmission des capteurs et l'activation du contrôle par l'API*. Ceci se fait grâce à l'application DJI Assistant 2 fournie et s'installe sur un ordinateur (Windows, iOS). Le *Baud Rate* est donc mis à 230400 et le reste sur 10Hz pour avoir une lecture des capteurs toutes les 100ms. Il serait possible de mettre plus rapide mais la consommation énergétique serait plus importante également.

L'OSDK tourne sur Ubuntu (16.0) mais peut également tourner sur Raspbian (OS pour le Raspberry Pi dérivé de Debian). Le code du OSDK tourne sur le Raspberry Pi et récupère les valeurs du drone via l'*UART*. Le SDK permet d'accéder aux données des capteurs de manière automatisée, ainsi que de déplacer le drone (décollage et atterrissage compris).

IV.3.3.2 Résultats

Une fois le OSDK installé et opérationnel sur le Raspberry Pi, j'ai pu exécuter les codes qui sont fournis en exemple. Sur la figure 14, l'exemple qui est montré ici est l'exemple où on interroge les capteurs de position (GPS), vitesse (centrale inertuelle), *RC Commands* correspond à la commande faite sur la télécommande (r : roll / p : pitch / y : yaw / thr : thrust).

```
Counter = 1950:
-----
Flight Status          = 1
Position      (LLA)    = 0, 0, 14.8651
RC Commands   (r/p/y/thr) = 0, 0, 0, 0
Velocity       (vx,vy,vz) = 0, 0, -0.00398795
Attitude Quaternion (w,x,y,z) = 0.996155, -0.00688457, -0.0168749, -0.0856936
Avoid obstacle data (down,front,right,back,left,up) =0, 0, 0, 0, 0, 0
-----
```

FIGURE 14 – Exemple fournis par DJI

IV.3.4 Drone : Positionnement dans la pièce

Afin de positionner le drone dans la pièce et étant donné que le GPS ne peut pas fonctionner dans la pièce, il est nécessaire d'utiliser un moyen externe. Nous avons ainsi choisi les cartes Decawaves MDEK1001 pour le positionnement. Tout d'abord, il faut créer le réseau avec les cartes. On crée donc un réseau grâce à l'application Android fournie (voir figure 15) puis on y assigne nos cartes Decawaves en paramétrant la position par rapport à une carte initiale qui sera positionnée en X = 0 ; Y = 0 ; Z = 0. On obtient donc avec 7 cartes ancrées et un tag ce que l'on peut

voir sur la figure 16 et le placement des cartes dans l'environnement est représenté par la figure 17.



FIGURE 15 – Composition du réseau

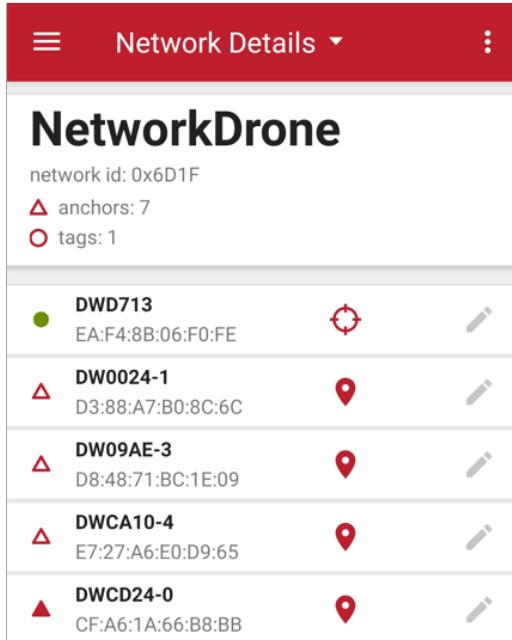


FIGURE 16 – Composition du réseau en détail

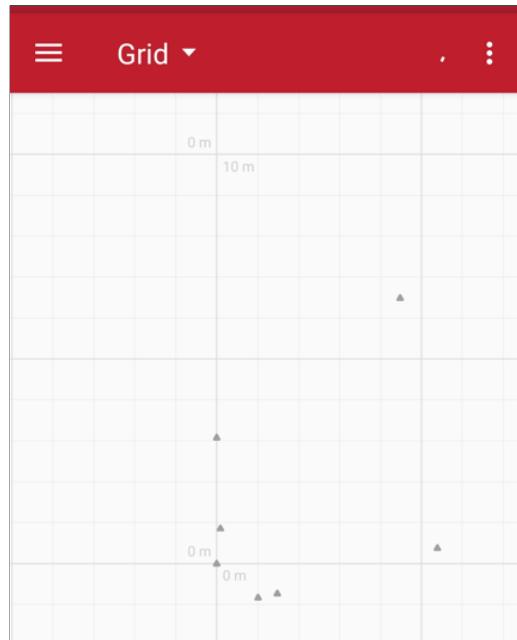


FIGURE 17 – Placement visuel des cartes dans l'espace

Une fois les cartes initialisées et le Raspberry Pi connecté à une carte Decawave, il est possible de récupérer les valeurs de position et de distance entre la carte du drone (tag) et les ancre positionnées.

Il y a deux solutions possibles. La première suffit d'aller sur l'application Android et de voir sur le grille la distance estimée du tag (Rappel : carte du drone). Mais cette solution n'est pas utile à long terme car ce que l'on désire c'est d'embarquer cette position et de diriger le drone en lui donnant cette position.

Ainsi, j'ai utilisé la seconde solution qui est d'interroger le tag via la connexion USB. Une fois les 4 cartes connectées et initialisées, il est possible via la commande Linux : `sudo minicom -D /dev/ttyACMO` d'accéder au moniteur série et d'exécuter

des commandes spécifiques à l'appareil connecté. Dans mon cas, les commandes utiles sont :

- "lep" montre la position sous forme : POS, positionX, positionY, positionZ, qualité du signal (en pourcentage)
- "les" montre la distance entre le tag et les 4 ancre les plus proches. Exemple :
 $1151[5.00,8.00,2.25] = 6.48$ 0CA8[0.00,8.00,2.25] = 6.51 111C[5.00,0.00,2.25] = 3.18 1150[0.00,0.00,2.25] = 3.16 le_us = 2576 est[2.57,1.98,1.68,100]

Une fois les valeurs de distances avec les cartes, nous avons pu établir des marges d'erreur et se rendre compte que la position estimée par Decawave (Méthode de Newton/Raphson) était avec une erreur de 10cms environ. J'ai donc voulu tester une méthode différente de celle implémentée dans la carte afin de vérifier s'il était possible d'avoir un meilleur résultat. La méthode de Decawave mettait une limitation dans le nombre de cartes qu'elle utilisait. En effet, il ne pouvait en aucun cas y avoir plus que 4 cartes Decawaves utilisées pour le calcul de la position. Or, normalement, plus le nombre de cartes est élevé et plus la précision est grande. C'est pourquoi, j'ai voulu voir avec la méthode de Bancroft qui prend un grand nombre de cartes ; tout en faisant le comparatif entre les deux méthodes pour voir si j'obtenais de meilleurs résultats avec la contrainte de temps que j'avais : avoir une réponse toute les 100ms maximum !

En exécution normale, le drone se déplace grâce aux données que calcule et lit le Raspberry Pi. Le programme qui tourne sur le Raspberry Pi est construit en multi thread. Un thread gère la communication avec le drone via le *SDK* et l'autre thread gère le positionnement et le déplacement via les cartes Decawave.

IV.3.4.1 Méthode de Bancroft

La méthode de Bancroft[13] est celle utilisée pour le positionnement GPS. Pour résoudre le problème de position de l'objet à localiser, il faut au minimum 3 valeurs de distance et une quatrième pour vérifier ou éliminer une deuxième solution qui serait possible à cause du passage 2D/3D.

Ainsi, pour un satellite, il est nécessaire de résoudre l'équation 1 suivante afin de trouver la position :

$$PR^j = \sqrt{(x^j - x)^2 + (y^j - y)^2 + (z^j - z)^2} + c\delta t \quad (1)$$

Si on prend une matrice B contenant chacun des paramètres et pseudo range de chaque satellite, et qu'on appelle des paramètres L, a, M et Λ tels que :

$$B = \begin{pmatrix} x^1 & y^1 & z^1 & PR^1 \\ x^2 & y^2 & z^2 & PR^2 \\ x^3 & y^3 & z^3 & PR^3 \\ x^4 & y^4 & z^4 & PR^4 \end{pmatrix}, L = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}, M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix},$$

$$\Lambda = \frac{1}{2} \langle \begin{bmatrix} r \\ c\delta t \end{bmatrix}, \begin{bmatrix} r \\ c\delta t \end{bmatrix} \rangle$$

On obtient donc une équation générale pour 4 satellites avec \mathbf{B}^{-1} comme étant l'inverse de la matrice \mathbf{B} :

$$\langle \mathbf{B}^{-1}, \mathbf{B}^{-1}\mathbf{L} \rangle \Lambda^2 + 2 [\langle \mathbf{B}^{-1}\mathbf{L}, \mathbf{B}^{-1}\mathbf{a} \rangle - 1] \Lambda + \langle \mathbf{B}^{-1}\mathbf{a}, \mathbf{B}^{-1}\mathbf{a} \rangle = 0 \quad (2)$$

L'équation 2 montre la formule générale pour calculer une position d'un point selon 4 satellites. Dans mon cas, les satellites seront les cartes Decawaves placées dans la pièce et le point à trouver est le drone. Il y a la notion de rayon de la Terre dont il faut s'affranchir.

Juste pour information, si l'on désire avoir une précision plus importante il est nécessaire d'augmenter le nombre de mesures et là on passe donc à une autre formule qui est :

$$\mathbf{B}^T \mathbf{a} - \mathbf{B}^T \mathbf{B} \mathbf{M} \begin{bmatrix} r \\ c\delta t \end{bmatrix} + \Lambda \mathbf{B}^T \mathbf{L} = 0 \quad (3)$$

Il est nécessaire de passer sur une autre formule car si il y a plus que 4 observations la matrice \mathbf{B} n'est plus carré. Il faut donc multiplier par la transposée de \mathbf{B} (\mathbf{B}^T) pour obtenir une solution.

A savoir que les équations 1, 2 et 3 sont des équations que je n'ai pas manipulé directement mais seulement par des librairies en C++ qui me facilitaient le travail.

A présent, si on applique cette méthode à mon cas de cartes Decawaves et un tag, on représente le système par le fait que les satellites sont remplacés par les cartes et le drone est le point dont on veut connaître la position.

Nous avons essayé avec 4 cartes au début et j'obtenais des résultats qui n'était pas très bon, de l'ordre d'un mètre de décallage avec des valeurs négatives parfois. J'ai donc regardé pour appliquer la méthode de Bancroft avec davantage de cartes et donc un calcul plus précis de la position. Mais au final, je me suis rendu compte qu'il y avait une limite logicielle dans le nombre de cartes auxquelles le drone pouvait s'adresser (voir figure 18). Cette limite étant de 4, il n'y avait pas de solution pour calculer plus précisément la position à partir de la commande du moniteur série "les".

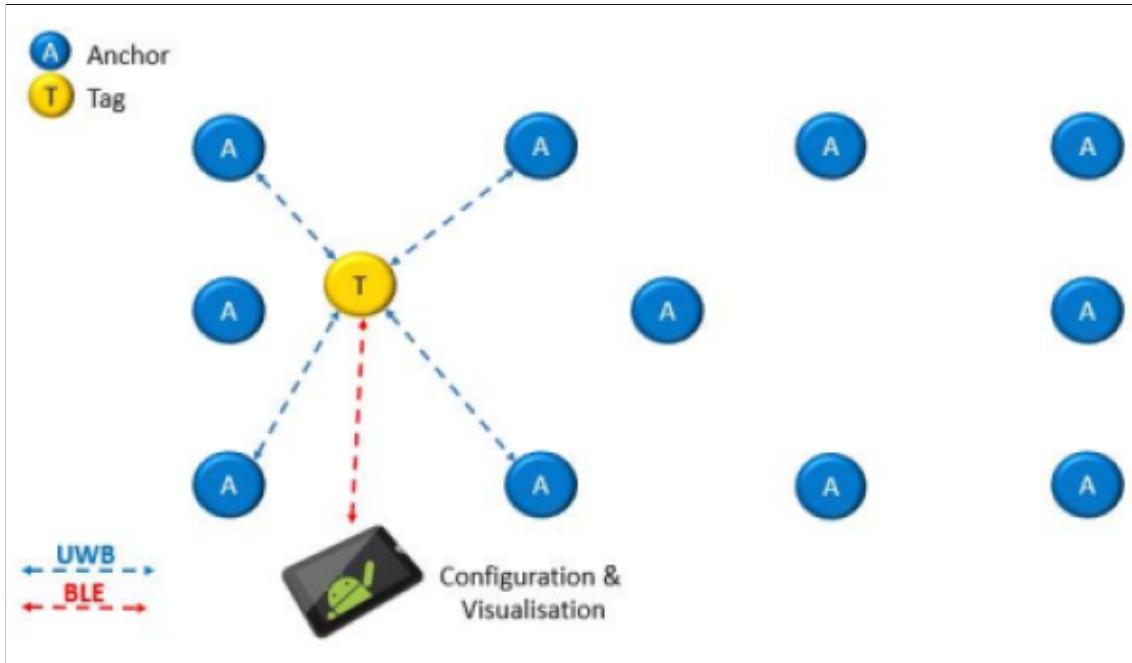


FIGURE 18 – Représentation d'une configuration avec 1 tag et 11 ancre

Pour tester la méthode de Bancroft, j'ai utilisé la librairie fournie par l'Université du Texas à Austin (Space and Geophysics Laboratory (SGL)) qui se nomme GPSTk[14]. Cette librairie est Open Source et j'ai donc pu récupérer seulement la partie qui m'intéressait car la librairie entière permet des calculs avec l'orbite des planètes, satellites, etc. Elle est très complète mais était trop complète pour mon usage, j'ai donc récupéré seulement les fichiers servants à la méthode de Bancroft et je les ai inclus dans mon Raspberry Pi pour la compilation et l'exécution du code.

Lors de ces essais, j'ai constaté que le positionnement des cartes et leur orientation jouaient un rôle assez important dans la précision et aussi dans la qualité du signal reçu par le drone et également des distances courtes entre chaque carte 'ancre' favorisaient de plus larges erreurs à cause du bruitage trop élevé.

IV.3.4.2 Méthode de Newton/Raphson

La méthode de Newton/Raphson est la méthode qui est probablement implémentée au sein des cartes Decawave par le constructeur. Il n'est pas possible de vérifier mais il est possible de faire des hypothèses notamment grâce au fait qu'il ne prenne que 4 cartes pour faire les calculs et jamais davantage.

Cette méthode, itérative, est efficace pour trouver numériquement une approximation d'une fonction.

Cette fonction appliquée dans l'environnement nous donne des résultats sans davantage de calculs avec une précision de l'ordre de 5 centimètres en moyenne si les

distances des cartes sont plus élevés que ce que j'avais au départ pendant mes essais.

Cette méthode étant simple à utiliser et très rapide car il est possible de l'avoir toutes les 100ms fait que je l'ai choisie par rapport à la méthode de Bancroft qui était très légère également mais avec des résultats très mauvais.

IV.3.4.3 Résultats

Pour présenter les différents résultats, je vais comparer les méthodes de Bancroft et celle de Newton/Raphson (Decawave). Les comparaisons se feront en terme de temps d'exécution pour voir si la méthode correspond aux exigences de temporalité demandée (période de 100ms), mais aussi avec la valeur de position calculée et l'écart par rapport à la réalité ainsi que la valeur moyenne et engin l'erreur moyenne calculée sur 1000 itérations.

Comme il est possible de voir sur la figure 19, le drone est positionné environ au milieu des ancre afin de tester. Le test se fait avec 7 ancre et un tag se trouvant sur le dessus du drone. Les valeurs de positions sous chaque ancre ou tag sont indiquées de cette façon : X, Y, Z. La distance entre l'ancre et le tag (trait en pointillé) est une distance mesurée avec un mètre et chaque position de drone est également mesurée au mètre par rapport à l'ancre de référence (ancre 0 en $[0 \ 0 \ 0]$).

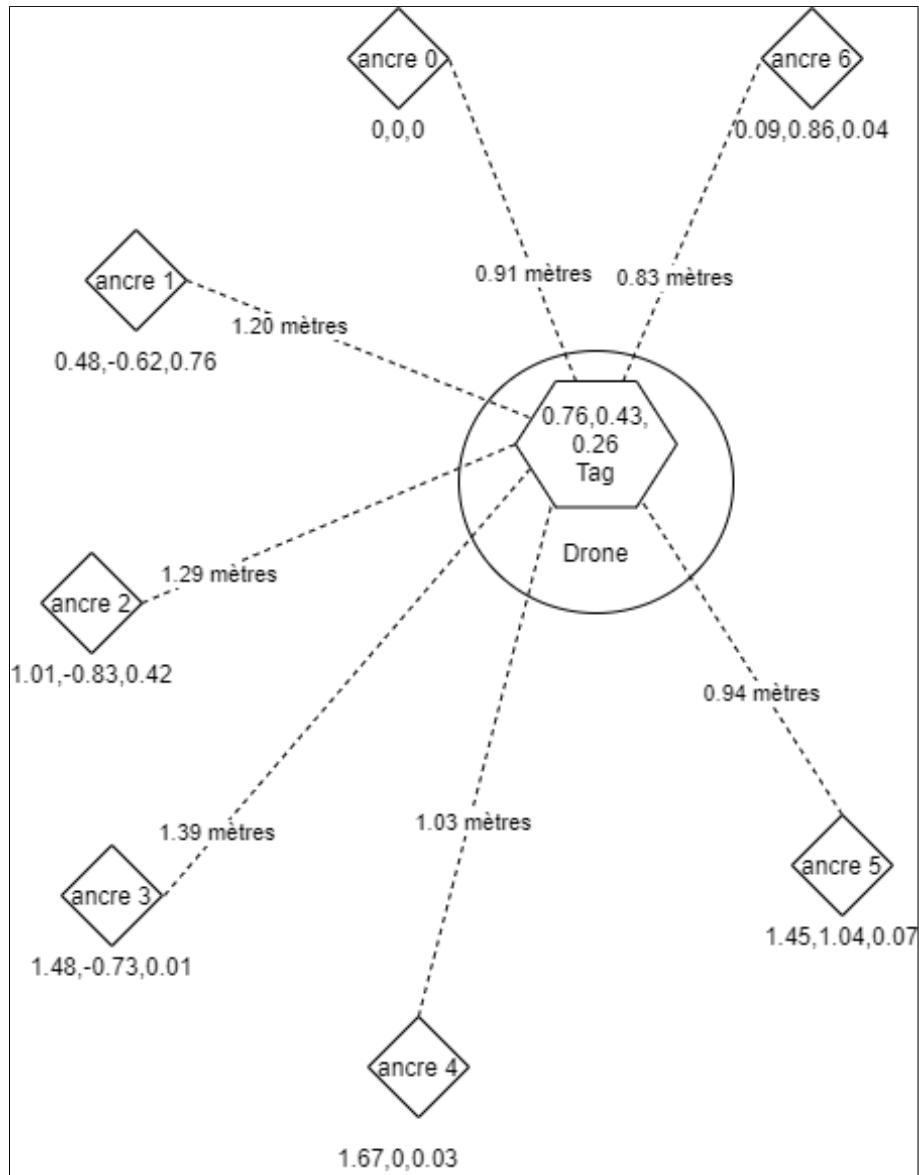


FIGURE 19 – Schéma représentant le positionnement du drone et des ancre avec distance mesurée

Les résultats présentés sur le tableau 1 montrent la comparaison de la valeur calculée par la méthode de Bancroft en utilisant 5 et 7 ancre actives par rapport à la réalité. Il est vrai comme indiqué précédemment que les cartes Decawaves ne prennent que 4 cartes en même temps, les résultats avec 5 et 7 sont possibles mais le tag ne prendra que 4 cartes, c'est juste qu'il aura le choix et donc choisira les plus proches ou d'autres selon la qualité du signal et donc on obtient des valeurs un peu plus précises.

On voit donc que le temps d'exécution est très bon, d'un facteur entre 100 et 200 environ. Les résultats du vecteur sont par contre très approximatif. On peut constater qu'en X la valeur est très correct à 8 centimètres pour les 5 ancre et 5

centimètres pour les 7 ancre. Par contre, là où le problème se pose c'est au niveau du Y et Z qui là peut atteindre une erreur de 1 mètre 20 ! Cette erreur est impossible à maintenir pour une application.

| Méthode | Temps d'exécution | Valeur du vecteur résultat |
|------------------|-------------------------|----------------------------|
| Réalité | / | 0.76 0.43 0.26 |
| Bancroft 7 ancre | entre 520 µs et 1200 µs | 0.71 0.80 0.78 |
| Bancroft 5 ancre | entre 450 µs et 1200 µs | 0.68 1.37 1.46 |

TABLE 1 – Résultats obtenus avec Bancroft

Les résultats du tableau 2 montrent la comparaison entre la réalité de la position du drone et la valeur calculée par Bancroft avec 4 ancre disponibles et Decawave avec 4 ancre également. Il est possible de voir que le temps d'exécution est très largement respecté pour les deux méthodes mais par contre la valeur du vecteur résultat est très différent selon les méthodes. La méthode de Bancroft prouve par ce tableau et ces résultats qu'elle n'est pas possible d'utilisation dans notre cas, peut-être car les distances sont trop petites. Pour avoir un ordre numérique de marge d'erreur, j'ai calculé la racine de l'erreur quadratique moyenne (RSME - Root Mean Square Error). La formule est :

$$err = \sqrt{(posX - droneX)^2 + (posY - droneY)^2 + (posZ - droneZ)^2} \quad (4)$$

Dans l'équation 4, la valeur **posXYZ** correspond à la position calculée par la méthode (Bancroft ou Newton avec Decawave) que ce soit en X, Y ou Z ; et **droneXYZ** correspond à la valeur réelle du drone en X, Y ou Z, cette valeur étant fixée manuellement dans le code pour les tests car le drone ne se déplaçait pas.

| Méthode | Temps d'exécution | Valeur du vecteur résultat | RSME |
|----------|-------------------|----------------------------|------|
| Réalité | / | 0.76 0.43 0.26 | / |
| Bancroft | $\cong 840 \mu s$ | 0.98 -0.11 0.35 | 0.34 |
| Decawave | $\cong 840 \mu s$ | 0.78 0.38 0.26 | 0.07 |

TABLE 2 – Résultats obtenus avec Bancroft et Decawave

Les résultats du tableau 3 montrent les valeurs moyennes obtenues ainsi que l'erreur moyenne sur 1000 itérations de boucle. Ce calcul montre définitivement que Bancroft n'est pas utilisable, une erreur moyenne de 0.88 en Y est énorme et ne peut pas être utilisée. Alors que l'erreur pour Decawave ne dépasse pas les 0.08, ce qui est très bon comme résultat. Surtout que dans la suite du stage, la hauteur de déplacement du drone n'était pas forcément prise en compte car on peut également

fixer le drone à une altitude de déplacement suffisamment haute pour éviter toute collision et le faire avancer en ligne droite sans devoir se repositionner en Z.

| Méthode | Valeurs moyennes | Erreurs moyennes sur X, Y, Z |
|----------|-----------------------------------|-------------------------------------|
| Réalité | / | / |
| Bancroft | X = 0.98 Y = -0.11 Z = 0.35 | X = 0.223 Y = 0.884 Z = 0.418 |
| Decawave | X = 0.77 Y = 0.38 Z = 0.26 | X = 0.014 Y = 0.068 Z = 0.084 |

TABLE 3 – Résultats moyens obtenus avec Bancroft et Decawave

IV.3.4.4 Problèmes rencontrés

Dans cette partie, le plus gros soucis a été la connexion série en C++. Il n'existe pas de librairie qui le fasse facilement comme en Python avec pyserial. Il a donc fallu comprendre le fonctionnement et recréer les structures nécessaires grâce au site Web *Linux Serial Ports Using C/C++*[15].

Le second problème a été de trouver la meilleure solution de positionnement du drone. Comme j'ai pu montrer dans les paragraphes précédents, la méthode de Decawave au départ nous paraissait pas très précise alors qu'au final nous l'avons choisie car parmi les 2 méthodes testées c'était la meilleure en tout.

IV.3.5 Drone : Déplacement dans la pièce

Le déplacement du drone a été choisie très simplement. La solution la plus directe c'est la ligne droite.

Au départ, j'ai cherché une fonction dans le *SDK* qui permettait de donner une trajectoire au drone de manière automatisée d'un point A à un point B. En partant du principe que le drone est orienté de la même façon tout le temps (voir figure 20) par exemple sur l'axe des X qui est fixé hypothétiquement sur l'axe Nord de la pièce.

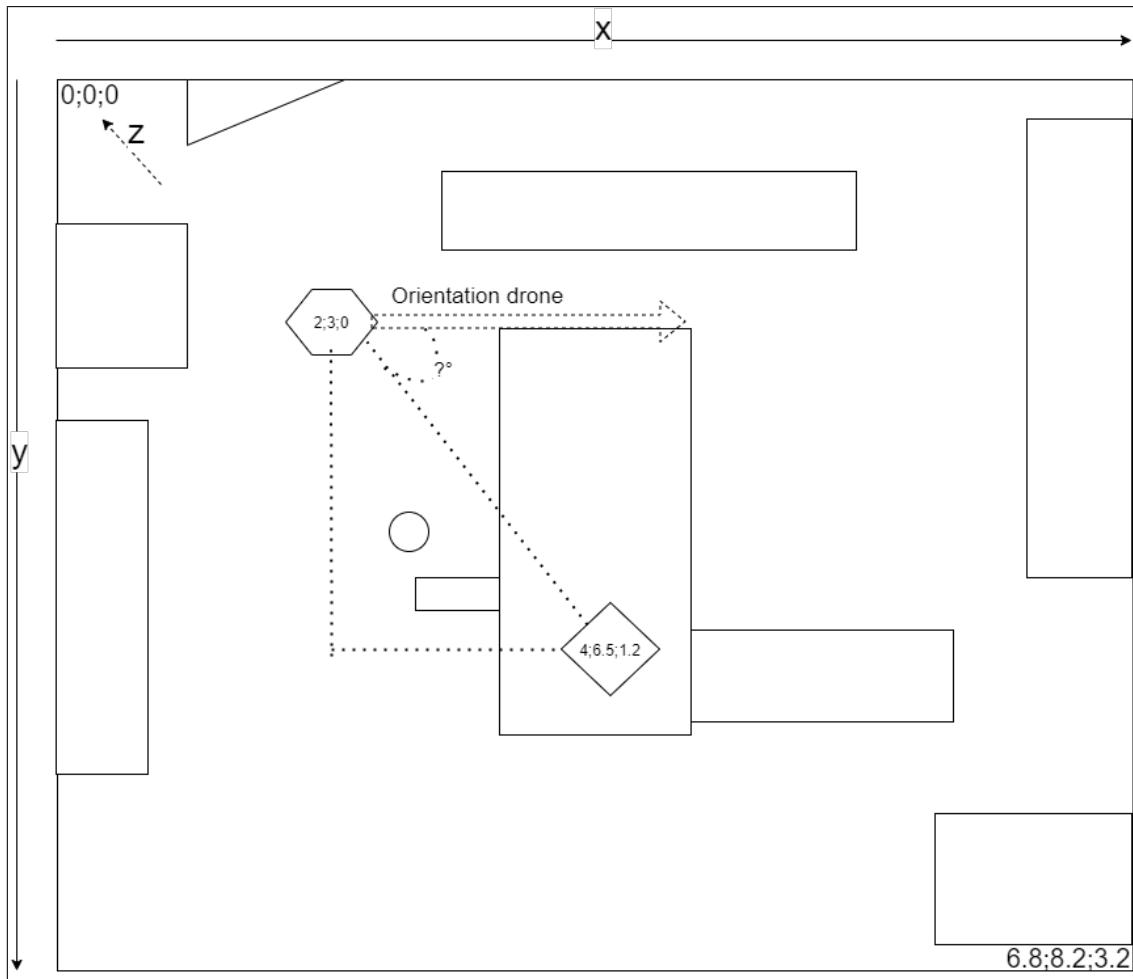


FIGURE 20 – Schéma représentant l'orientation du drone par rapport au point d'arrivée

J'ai ainsi trouvé la fonction fournie (voir le bloc de code 3) par le constructeur qui permet un déplacement selon une distance en X, Y et Z avec possibilité de lui indiquer un angle de déplacement. Cette fonction correspondait parfaitement à mes attentes mais le soucis était de trouver l'angle de déplacement comme représenté sur la figure 20. On connaît la position du drone, on connaît la position du capteur mais l'angle de direction du drone est à déterminer sauf si on part de l'hypothèse que le drone est dans une direction fixe.

Concernant l'axe Z, j'ai fait le choix, pendant le déplacement jusqu'au capteur, que le drone se positionne à une altitude donnée comme par exemple 2 mètres puis se tient sur cette altitude jusqu'au point et là réévalue l'altitude nécessaire pour la prise de la photo.

```
1 bool moveByPositionOffset(DJI::OSDK::Vehicle *vehicle ,  
2     float xOffsetDesired ,  
3     float yOffsetDesired ,  
4     float zOffsetDesired ,  
5     float yawDesired ,  
6     float posThresholdInM = 0.5 ,  
7     float yawThresholdInDeg = 1.0) ;
```

Listing 3 – Fonction de déplacement du drone

Comme vu dans le paragraphe précédent, la fonction de déplacement demande un angle afin de s'orienter dans la direction. Le drone a permis ses capteurs une boussole. Mais cette boussole doit être initialisée au départ du drone avec vue sur le ciel. Dans notre pièce, l'initialisation était impossible, ainsi à partir de ce moment il me fallait lancer le drone et l'initialiser dehors avec vue sur le ciel pour le calibrer.

Dans la version du *SDK* que j'utilise, il n'y a aucune fonction existante afin d'utiliser directement la boussole et la valeur renvoyée par le capteur. Par conséquence, je me suis orienté pour trouver une autre solution, un autre capteur sur le drone pour répondre à mes besoins. J'ai trouvé le magnétomètre et bien que la fonction soit notée seulement pour les drones *M210V2* et *M300*, la fonction fonctionne quand même sur mon drone. Malheureusement, les mesures étaient impossibles à deviner et la logique était compliquée à saisir sans avoir de documentation sur le fonctionnement précis de la fonction.

Après davantage de recherches et d'explorations au sein de la documentation sommaire fournie ainsi que l'exploration directe du code à ma disposition dans les fichiers sources du *SDK*, j'ai trouvé une donnée qui correspond à une mesure de boussole.

Cette mesure s'échelonnait de 0 à 0.9999 si on allait du Nord au Sud en passant par l'Est. Et si le drone allait du Nord au Sud en passant par l'Ouest, les mesures étaient de 0 à -0.9999. J'ai donc réalisé un étalonnage le plus précisément possible par rapport aux mesures que le drone nous donnait. Les résultats de ce relevé sont visibles sur la figure 21 représentée par une rose des vents.

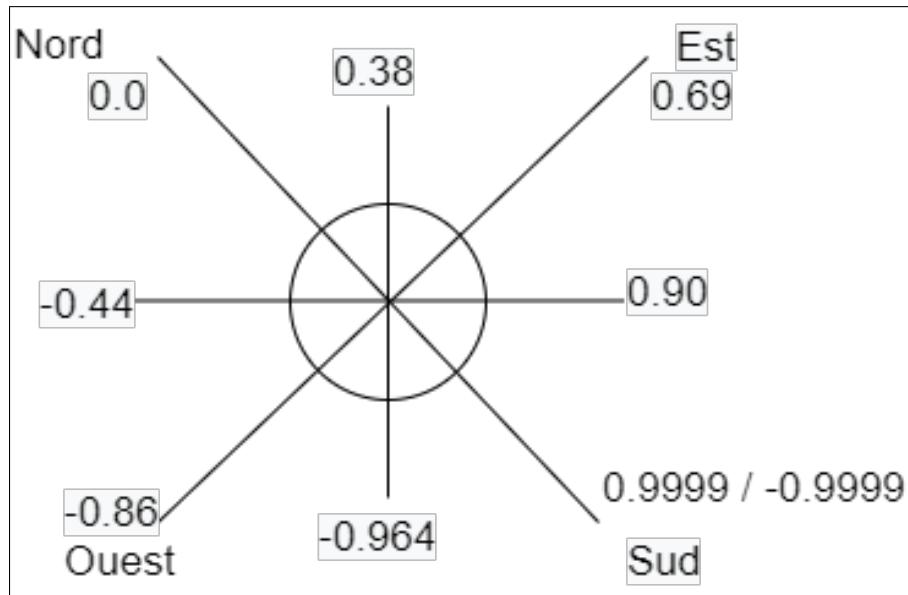


FIGURE 21 – Schéma représentant les données du capteur selon l'angle du drone

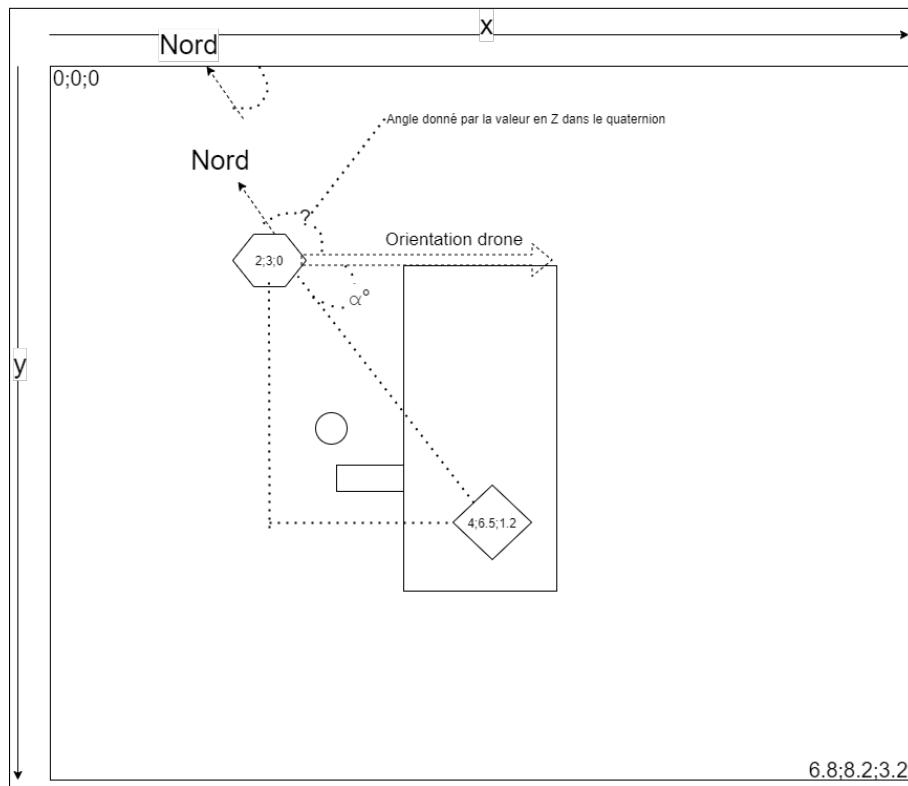


FIGURE 22 – Schéma représentant l'angle à déterminer selon l'orientation du drone

Ainsi, pour finir le déplacement du drone d'un point A à un point B, je suis resté sur cette valeur qui n'est pas la plus précise mais qui est fiable malgré le contexte

très défavorable de la pièce. Pour calibrer et étalonner la valeur reçue par rapport au drone, j'ai utilisé une fonction que j'ai récupéré d'Arduino qui se nomme *map()* et qui prend une valeur en paramètre à réétalonner sur une autre échelle.

```

1 double map(double x, double in_min, double in_max, double out_min,
2           double out_max){
3     return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
4            out_min;
5 }
6 float angleReel = map(angle, 0.64, 0.90, 90, 135);

```

Listing 4 – Exemple fonction map

Comme on peut le voir sur le bloc de code 4, je pars d'une variable *angle* qui prend une valeur entre 0.64 et 0.90 et je transforme cette valeur vers la nouvelle plage de valeur qui est entre 90 et 135 degrés.

IV.3.6 Conclusion partielle

L'objectif de cette partie était de pouvoir accéder au drone depuis un ordinateur embarqué qui serait installé sur le drone. Cet ordinateur devait pouvoir gérer le déplacement, positionnement ainsi que la prise d'image ou l'analyse des images prises. L'ordinateur a donc été choisi avec ces contraintes. Un deuxième objectif était de pouvoir positionner le drone à l'intérieur de la pièce ainsi que de le faire se déplacer automatiquement d'un point de départ à un point d'arrivée avec le retour au point de départ ou un point qui serait déterminé comme base du drone.

Pour réaliser tous ces objectifs, j'ai, grâce au Raspberry Pi, pu intégrer le *SDK* du drone fourni par le constructeur DJI. Ce *SDK* me permettait d'accéder à l'autopilote du drone et de gérer le déplacement du drone. Enfin, le positionnement s'est fait grâce à un moyen externe au drone. Les cartes Decawaves avec une communication *UWB* m'ont permis de placer le drone de manière assez précise (erreur quadratique moyenne de 0.07) avec une déviation de seulement quelques centimètres.

IV.4 Troisième partie : Le réseau de neurones

IV.4.1 IA : Généralités

Le laboratoire possède un serveur à disposition du personnel afin de lancer des programmes très lourd comme par exemple des apprentissages de réseaux de neurones. Le serveur possède 64 Gigaoctets de RAM, une carte graphie Nvidia Quadro P4000 et un processeur Intel Xeon Gold 5122 à 3.6GHZ et possédant 8 coeurs / 16 threads.

L'objectif de cette partie était de choisir quelques anomalies possibles sur la plateforme et d'essayer de faire apprendre à un réseau de neurones qui serait implanté

sur le Raspberry Pi à reconnaître ces anomalies afin de les détecter et de traiter l'anomalie selon le degré de sévérité.

Les anomalies qui ont été choisies sont une absence de tube sur la palette et une absence de bouchon sur la palette. Elles ont été choisies car elles sont faciles à reproduire et facilement photographiable en grande quantité et sous plusieurs angles.

IV.4.2 IA : Dataset

Tout d'abord, afin de faire apprendre les anomalies à un réseau de neurones il a fallu prendre une grande quantité de photos par rapport aux anomalies que je cherche à détecter. C'est-à-dire, des photos de la palette avec un tube en plastique transparent comme il est possible de voir sur l'image 23 et des photos de la palette avec le bouchon.

Le réseau de neurones choisi pour les expérimentations a été Yolo (You Only Live Once)[16] dans sa version 5. Je développerai davantage le choix de ce réseau dans la partie IV.4.4.

Le dataset est donc composé de 25 images de bouchons, 25 images de tube transparent et 25 images de tube avec un papier coloré pour mettre en évidence le tube (voir image 24).

Les images ont été prises grâce à la Caméra V2 du Raspberry Pi qui avait été acheté auparavant. La caméra était fixée sur le Raspberry Pi fixé lui même sur le drone. Les images étaient ensuite prises via un script Bash selon un modèle donné.

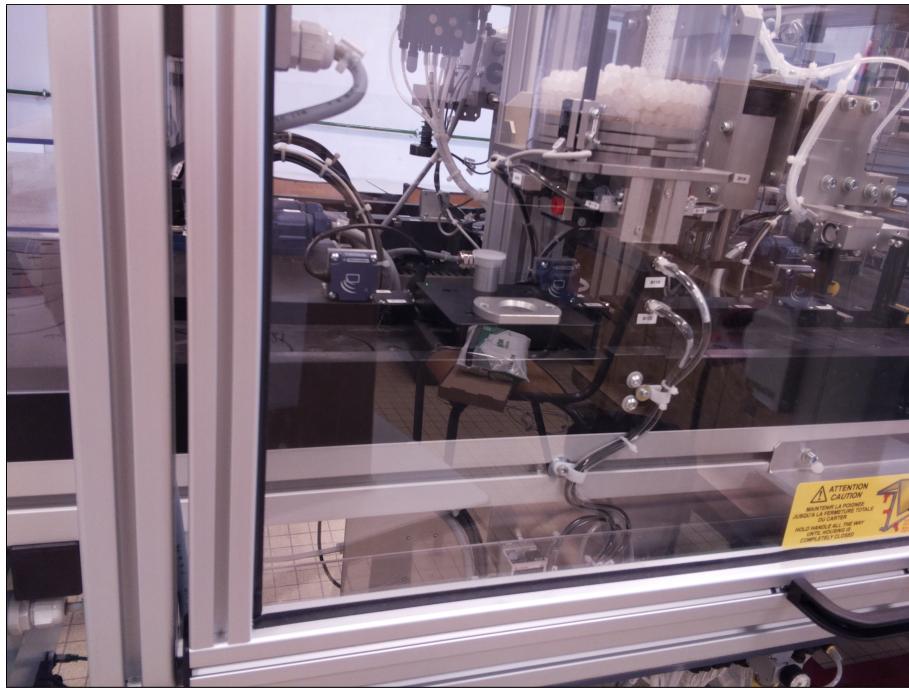


FIGURE 23 – Exemple d'image : Anomalie Bouchon



FIGURE 24 – Exemple d'image : Anomalie Tube coloré

IV.4.3 IA : Labellisation des images

Une fois le dataset fait, il est nécessaire de labelliser les images. C'est-à-dire de renseigner les objets que l'on veut mettre en évidence sur l'image. Dans mon cas, les deux objets qui étaient à mettre en évidence étaient le tube sur la palette et le bouchon sur la palette.

Afin de labelliser ce dataset, soit 75 images, j'ai utilisé un logiciel libre de droit, gratuit et écrit en python qui se nomme *labelImg*. Il est disponible sur GitHub[17]. L'interface permet de choisir le répertoire de travail, de sélectionner entre le réseau de neurones Yolo et PascalVOC, créer le rectangle pour indiquer où se trouve l'objet à montrer et ensuite on peut choisir la classe de l'objet dans une liste ou ajouter un objet dans cette liste. Il est très simple d'utilisation et m'a permis de labelliser les 75 images en environ 20 minutes.



FIGURE 25 – Interface graphique de labelImg

Le fichier créé lors de la labellisation donne juste une ligne sous la forme : *classe centreX centreY largeur hauteur*, ce qui donne par exemple pour la figure 26 quelque chose comme *0 0.48 0.63 0.69 0.71* avec 0 comme étant la classe Zidane. La classe est représenté par un chiffre (0, 1, ...) selon le nombre de classes dans le dataset.

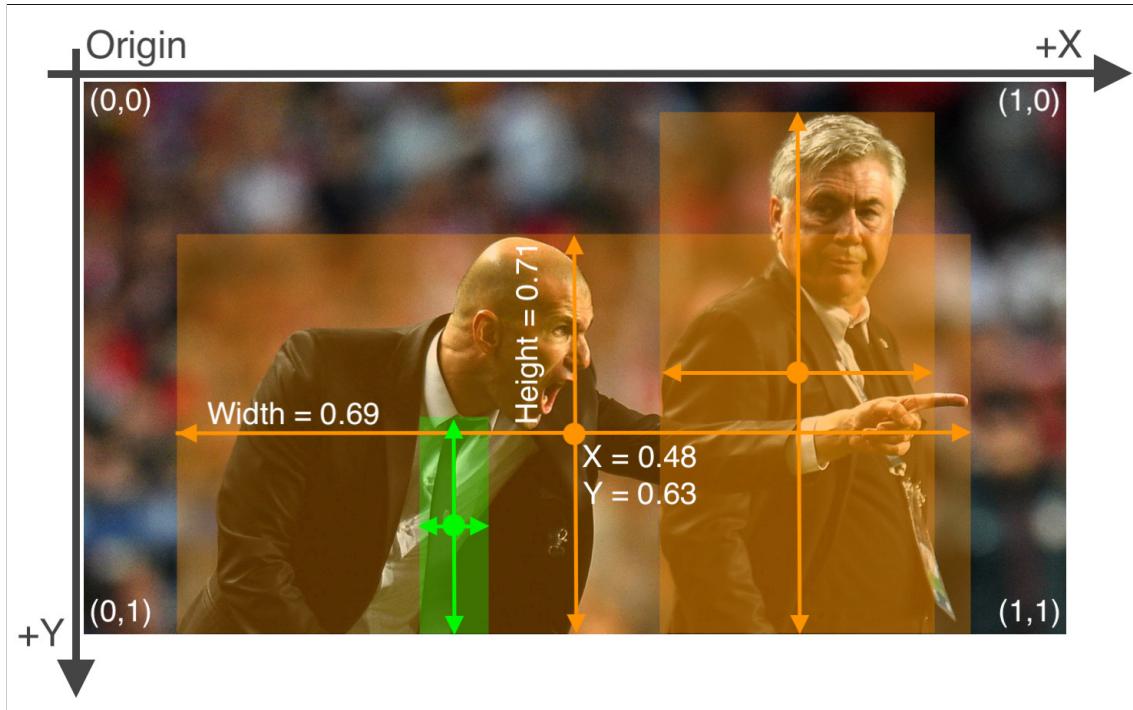


FIGURE 26 – Exemple de labellisation avec labelImg

IV.4.4 IA : YoloV5

IV.4.4.1 Un réseau de neurones

YoloV5 est un système de détection d’objets en temps réel basé sur le dataset COCO. Il est très simple à comprendre et à utiliser, et permet d’avoir des résultats très rapidement pour un dataset donné. Il existe de nombreux tutoriels et exemples sur le net sur lesquels il est possible de s’appuyer afin de démarrer au mieux. Une fois qu’un modèle est entraîné, il est possible de détecter les objets sur une vidéo en temps réel. C’est une des raisons pour lesquelles il a été choisi à la place d’autres réseaux existants.

IV.4.4.2 Utilisation de Yolo

Une fois le dataset labellisé, il est nécessaire de mettre en ordre les fichiers afin que Yolo puisse apprendre. Il faut donc créer des répertoires afin de séparer les images et les labels (répertoires images/ et labels/) puis de mettre des sous dossiers pour les entraînements (train), validation et tests. Une fois ceci fait, dans le dossier de Yolo, il est nécessaire de créer un fichier YAML (*Yet Another Markup Language*) qui contiendra les chemins d’accès aux dossiers précédents ainsi que les noms et le nombre de classes.

Nous avons donc à présent un dossier labels et un dossier images, chacun contenant des sous-dossiers avec minimum un dossier d'entraînement et de validation (pouvant être le même que l'entraînement) et facultativement un dossier de test. Et les images et labels rangés dans les dossiers correspondants et rangés de manière aléatoire. J'avais 75 images donc 25 bouchons, 25 tubes et 25 tubes colorés ; je les ai donc séparé en 17 bouchons, tube et tube coloré en entraînement (51 images) et le reste (24 images) en tests. Le dossier de validation est le même que l'entraînement.

IV.4.4.3 Apprentissage

A présent, pour l'apprentissage du modèle par le réseau de neurones, il est nécessaire tout d'abord de choisir le type de modèle. Il y a 4 modèles disponibles : le S, M, L et X.

J'ai éliminé d'office le modèle S car il donnait de résultats trop médiocre bien qu'il soit plus rapide. Le modèle X était le meilleur mais avec une différence très petite par rapport au modèle L pour un temps d'exécution 60% plus lent. Le choix se posait donc entre le modèle L et le modèle M, mais je me suis orienté vers le modèle L car il donnait de meilleurs résultats que le M pour une différence de temps minimale.

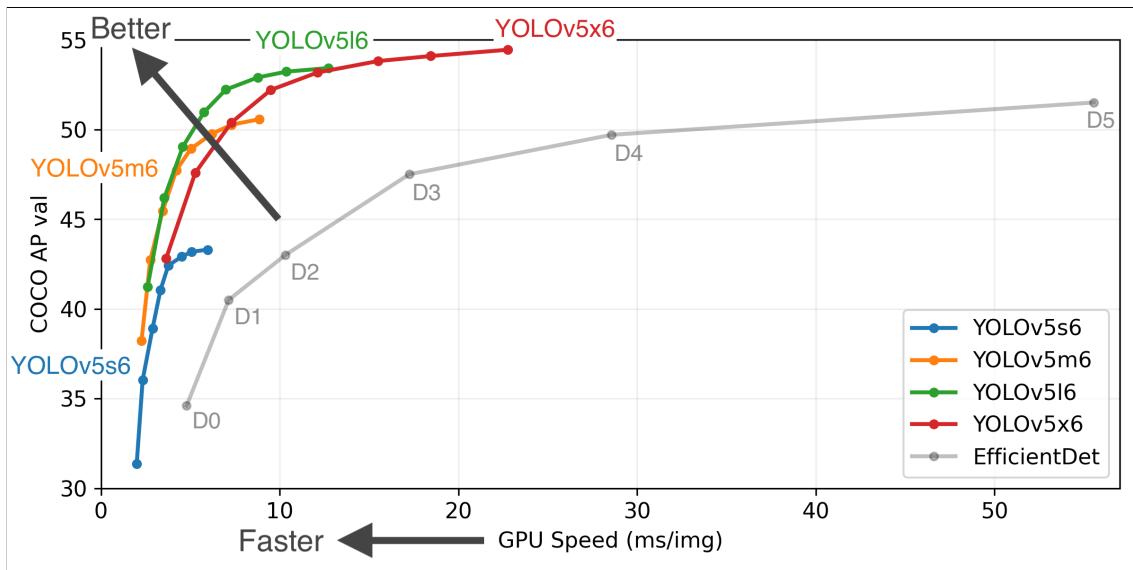


FIGURE 27 – Comparaison des modèles disponibles

Comme il est possible de voir sur la figure 28, les valeurs veulent dire que pour le modèle S pour l'exécution et l'apprentissage du dataset COCO, il fallait 14MB en float point 16 (FP16) de mémoire, 2.0ms d'exécution sur une carte graphique Nvidia Tesla V100 en moyenne sur 5000 images et enfin, 37.2mAP (Mean Average Precision) qui est calculé en prenant la moyenne des moyennes des précisions sur toutes les classes du dataset COCO.

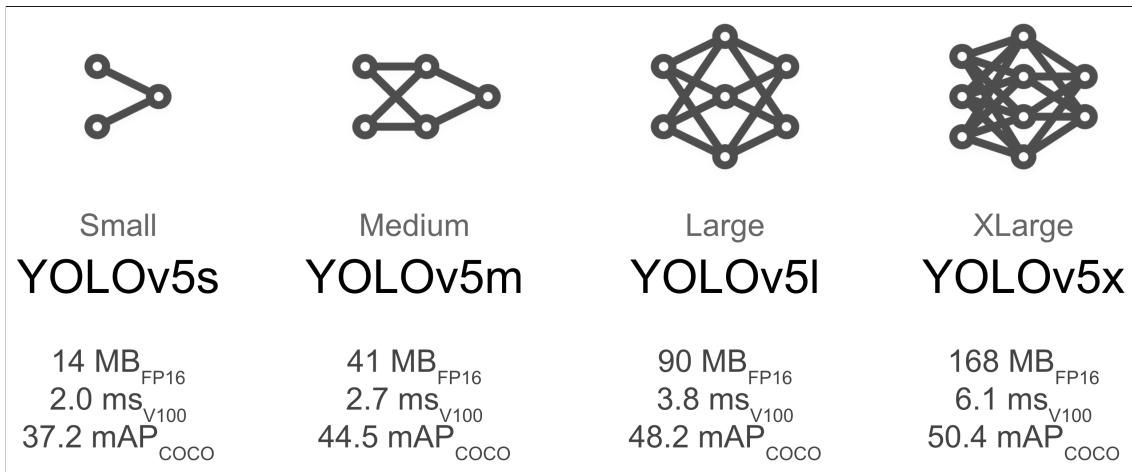


FIGURE 28 – Modèles disponibles pour YoloV5

Enfin, l'apprentissage par le réseau de neurones du dataset se fait via la commande : `python train.py -img 640 -batch 1 -epochs 100 -data plateforme.yaml -weights yolov5l.pt -cache -nosave`.

- img : permet de déterminer la taille des images, plus le chiffre est grand et plus le temps d'apprentissage sera élevé
- batch : le nombre d'images que le réseau de neurones prend à chaque propagation dans le réseau
- epochs : le nombre d'apprentissage. Pour les images et vidéos ce chiffre doit être grand (supérieur à 100, parfois même conseillé à 500/1000 si besoin)
- data : le fichier d'accès au dataset en format YAML
- weights : le modèle choisi

IV.4.4.4 Résultats

La commande d'apprentissage mise dans la sous-partie précédente a mis environ 3h30 à s'exécuter avec 1 batch et 100 epochs sur 51 images de taille 640.

A la fin de l'apprentissage, j'ai ainsi obtenu la matrice de confusion (voir figure 29) ainsi que des images prédites (voir figure 30) par l'algorithme.

On voit sur la figure 29 que l'algorithme apprend parfaitement les bouchons et les tubes, il détecte à 100% les images du dataset. La valeur du background FP est un élément de l'arrière plan d'une image qui a été classifié comme un tube lors de l'apprentissage. Il est impossible de voir les images incriminées pour plus de détails. Autrement, les images prédites sont très bonnes que ce soit sur les bouchons et les tubes, la prédiction est parfaite sur les images de tests que j'ai.

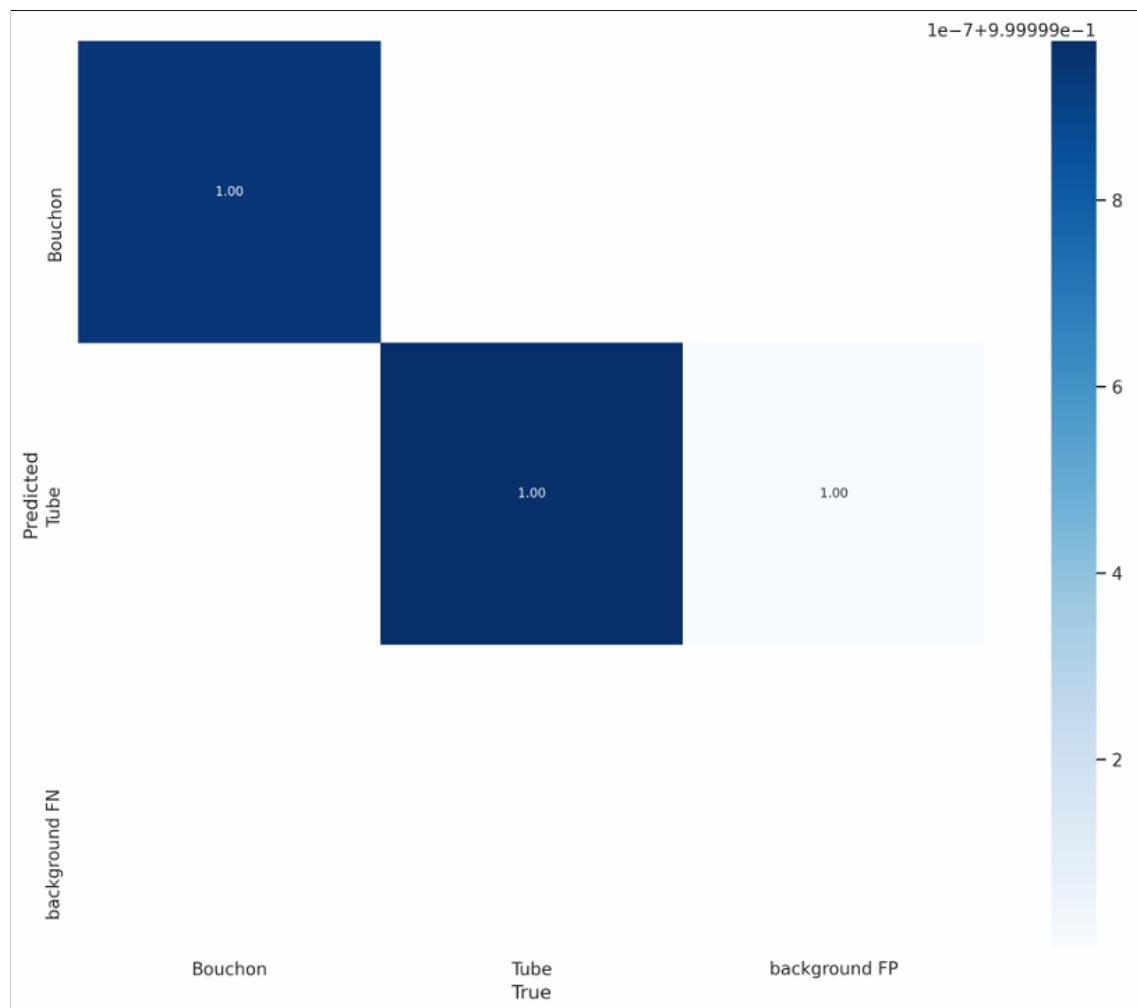


FIGURE 29 – Matrice de confusion

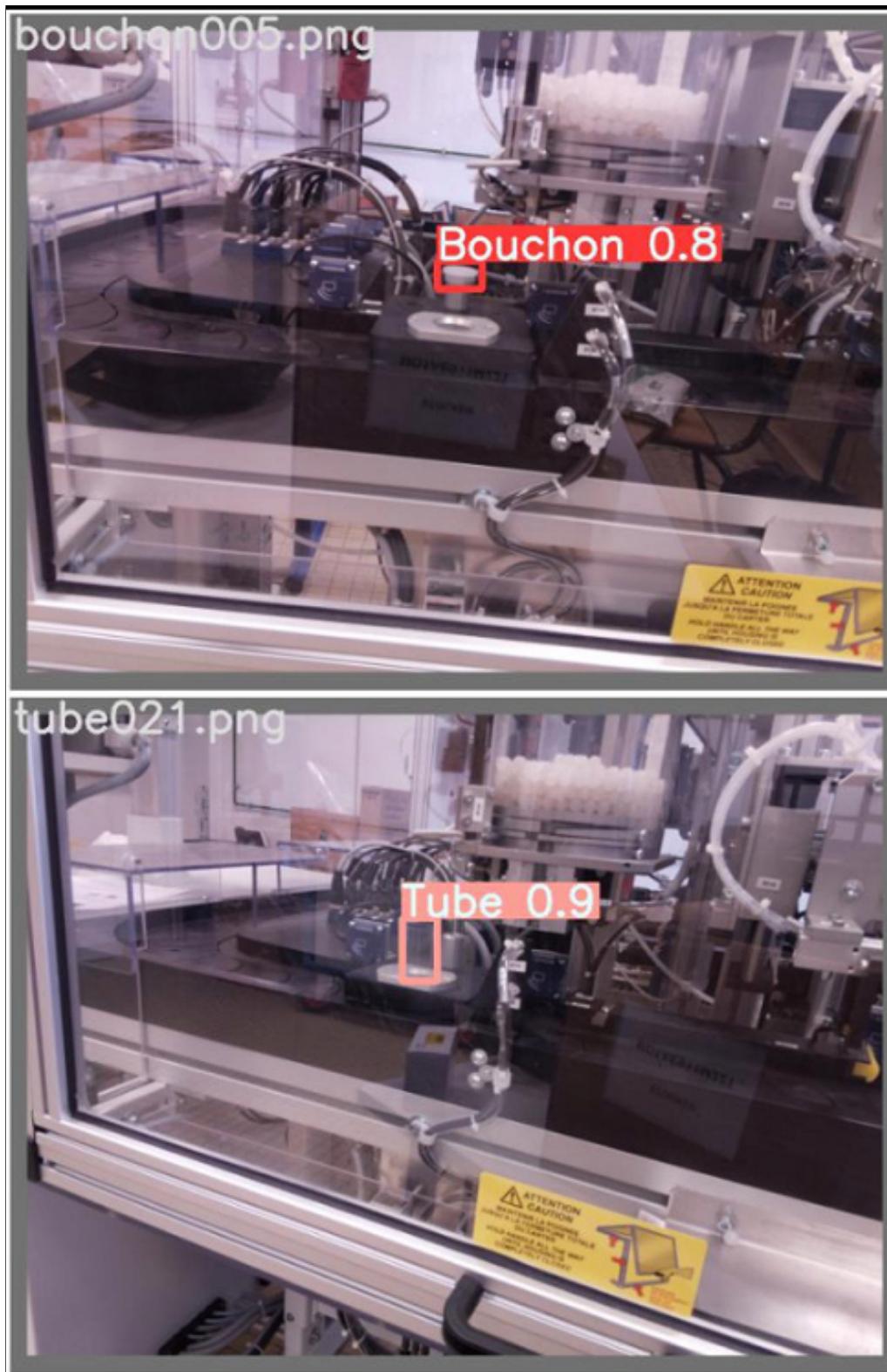


FIGURE 30 – Exemple de prédiction faite par l'apprentissage

IV.4.5 Conclusion partielle

L'objectif de cette partie était l'apprentissage d'un dataset fait à partir d'anomalies possibles sur la plateforme. L'apprentissage devait se faire par un réseau de neurones qui pouvait ensuite être intégré et embarqué sur le Raspberry Pi afin qu'il reconnaisse une anomalie ou non si un capteur de la plateforme détecte un problème.

J'ai, ainsi, grâce au logiciel *labelImg* et au réseau de neurones Yolo, pu réaliser cette étape. Les résultats sont excellents et peuvent à présent être utilisés pour intégrer le réseau généré sur le drone et le Raspberry Pi afin que le drone puisse intervenir automatiquement en cas de panne sur la plateforme. Bien sûr, pour le moment, il est capable de détecter seulement la présence ou non d'un tube et d'un bouchon sur la palette ; mais la base est présente et peut être utilisée pour aller plus loin sur les anomalies et les pannes possibles sur la plateforme entière car je me suis concentré sur la partie gauche de la plateforme (l'autre module étant en panne pour le moment).

V Conclusion du stage

Le but de ce stage était d'établir une coopération entre les dispositifs présents tels que la plateforme de production, les automates de cette plateforme, le drone et le serveur. La coopération devait mener à une surveillance active des composants en analysant les possibles pannes détectées par le système.

La première tâche du stage a été de me familiariser avec l'environnement. J'ai ainsi pu voir la plateforme industrielle en fonctionnement partiel. Et j'ai également pu découvrir le système de communication entre le Client ModBus et la plateforme faisant office de Serveur Modbus. Une fois ceci fait, j'ai créé un programme *Java* me permettant de récupérer les valeurs des registres des automates de la plateforme. Ces valeurs sont lues puis enregistrées dans une base de données MS SQL Server 2014 hébergée sur l'ordinateur Client. Ces données doivent servir plus tard pour le drone.

Dans un second temps, j'ai pris en main le drone, et relié le drone à un ordinateur externe embarqué sur le drone lui-même. Cet ordinateur embarqué, un Raspberry Pi 3B+, me permet d'exécuter le *SDK* fourni par le constructeur et d'accéder aux données des capteurs du drone ainsi que d'avoir une liaison avec l'autopilote du drone afin de le contrôler de manière automatique et donc d'exécuter des déplacements entre un point de départ et un point d'arrivée.

Puis enfin, dans un dernier temps, je me suis penché sur l'apprentissage par un réseau de neurones, YoloV5, de deux types d'anomalies que j'ai choisi de par leurs possibilités d'être reproduites facilement ainsi que par le fait qu'il est simple de les détecter. J'ai donc créé mon dataset en prenant seulement des images avec la caméra du Raspberry Pi des anomalies que je cherche à détecter. Puis, j'ai entraîné le réseau sur serveur et non pas sur le Raspberry Pi car cela aurait été vraiment très long à cause des petites performances de calcul par rapport à ce qui est demandé pour

un apprentissage de réseau de neurones. A présent, le réseau est généré et il est possible de l'exécuter sur le Raspberry Pi afin qu'il photographie la plateforme et détecte une anomalie par rapport à ce qu'il a appris dans cet apprentissage. Je n'ai pas eu le temps malheureusement de finir le projet. Il me reste seulement à intégrer le réseau créé par Yolo au Raspberry Pi et tester l'ensemble du code en condition réelle d'utilisation dans la pièce en prenant une photo et en voyant en temps réel s'il détecte la présence d'un tube ou d'un bouchon sur la plateforme et à plusieurs endroits du circuit si cela est possible !

J'ai beaucoup apprécié de stage du fait de son contenu riche et diversifié. J'ai pu découvrir le fonctionnement d'un automate ainsi que du protocole de communication ModBus. Mais également, découvrir le fonctionnement d'un drone et récupérer les données de celui-ci bien qu'il ne volait pas en intérieur (trop dangereux à cause des hélices). Enfin, la découverte du fonctionnement d'un réseau de neurones m'a beaucoup intéressé car jusque là je n'avais jamais eu la possibilité de créer moi-même un réseau et de le faire apprendre sur une machine et tout cela très facilement et en peu de temps.

Ce stage dans le monde de la Recherche me conforte dans mon souhait de continuer sur cette voie et d'accéder à une Thèse de Doctorat pour les années à suivre.

Annexes

Annexe 1 - Raspberry Pi

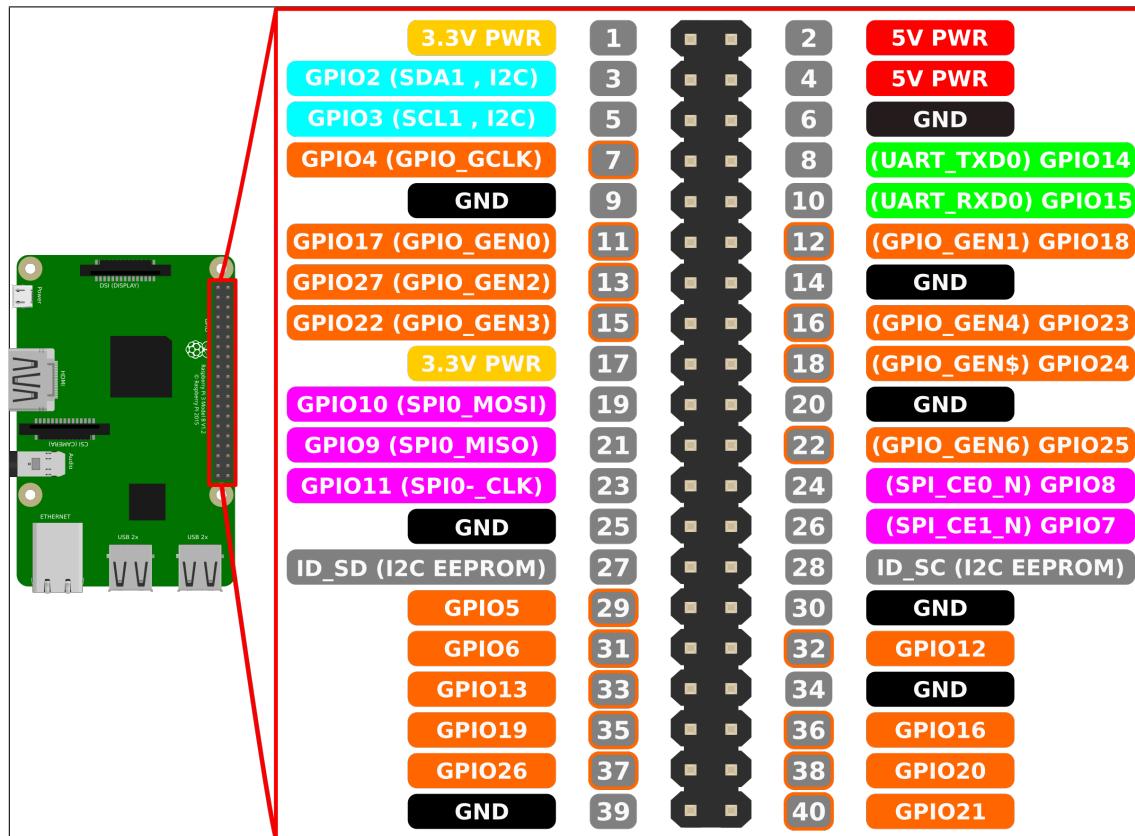


FIGURE 31 – Détails des pins GPIO du Raspberry Pi 3 B+

Les pins qui m'ont été utiles étaient les pins 6 pour le GND, 8 et 10 pour le UART_RX et UART_TX.

Annexe 2 - Code

```

1  public int connexion(){
2      if(master == null) {
3          try {
4              master = new ModbusTCPMaster(addr);
5              master.connect();
6              requete.connexion();
7          } catch (Exception e) {}
8          System.out.println(master);
9          if(master != null) return 0;
10         else return 1;
11     }
12     return 2;
13 }
```

Listing 5 – Connexion à l'automate

```

1  public void readVar() throws SQLException {
2      int n = 0;
3      while(n < nbRep){
4          if(typeReg == 'W')readAddrW();
5          else if(typeReg == 'M')readAddrM();
6
7          if((n + 1) != nbRep) {
8              try {
9                  Thread.sleep(timer);
10             } catch (InterruptedException e) {}
11         }
12         n++;
13     }
14     String tmp = "";
15     if(typeReg == 'W')tmp = "MW";
16     else if(typeReg == 'M') tmp = "M";
17     requete.select("*", "WHERE registerName = '" + tmp + addrReg + "'")
18 }
```

Listing 6 – Séparation des appels

```

1  /**
2   * @description Permet de lire les variables de type %MW sur l'automate
3   * @throws SQLException
4   */
5  public static void readAddrW() throws SQLException {
6      try {
7          InputRegister[] inReg = master.readInputRegisters(addrReg,
sizeReg);
```

```

8     if(inReg == null) {
9         System.out.println("Problème de lecture");
10        System.exit(1);
11    }
12    for(int i = 0; i < inReg.length; i++) {
13        System.out.print(inReg[i] + " ");
14    }
15    System.out.println();
16
17    requete.insert("MW"+ addrReg, inReg[0]);
18
19 } catch (ModbusException e) {
20     e.printStackTrace();
21     System.exit(1);
22 }
23 }
24
25 /**
26 * @description Permet de lire les variables de type %M sur l'automate
27 * @throws SQLException
28 */
29 public static void readAddrM() throws SQLException {
30     try {
31         BitVector inReg = master.readInputDiscretes(addrReg, sizeReg);
32         if(inReg == null) {
33             System.out.println("Problème de lecture");
34             System.exit(1);
35         }
36         byte[] tab = inReg.getBytes();
37         System.out.println(tab[0]);
38
39         requete.insert("M"+ addrReg, tab[0]);
40     } catch (ModbusException e) {
41         e.printStackTrace();
42         System.exit(1);
43     }
44 }
45 }
```

Listing 7 – Lecture des registres

```

1 Adresse de l'automate (défaut : 192.168.1.4) :
2
3 Temps entre chaque lecture (défaut : 5000ms) :
4 2000
5 Nombre de répétitions de la lecture (défaut : 1) :
6 2
7 Adresse du registre à lire :
8 %M103
9 Taille du registre à lire (défaut : 1) :
10 1
```

```

11 Données enregistrées :
12 192.168.1.4 | 2000 | 2 | 103 | 1
13 Si c'est OK, entrez 'ok'
14 ok
15 -----
16
17 Client connecté
18 === Contenu des registres ===
19 0
20 1
21 Client déconnecté

```

Listing 8 – Exemple d'exécution sur un registre de type %M

```

1 Adresse de l'automate (défaut : 192.168.1.4) :
2
3 Temps entre chaque lecture (défaut : 5000ms) :
4 10000
5 Nombre de répétitions de la lecture (défaut : 1) :
6 3
7 Adresse du registre à lire :
8 %MW150
9 Taille du registre à lire (défaut : 1) :
10 4
11 Données enregistrées :
12 192.168.1.4 | 1000 | 3 | 150 | 4
13 Si c'est OK, entrez 'ok'
14 ok
15 -----
16
17 Client connecté
18 === Contenu des registres ===
19 6400 0 10 38
20 12800 0 10 38
21 19200 0 10 38
22 Client déconnecté

```

Listing 9 – Exemple d'exécution sur un registre de type %MW

```

1 /**
2 * @description Permet de créer une requête d'insertion dans la base de
3 * données
4 * @param name : nom du registre
5 * @param value : valeur du registre
6 * @throws SQLException
7 */
8 public void insert(String name, Object value) throws SQLException{
9     String sql = "INSERT sensorValue (registerName, registerValue)
VALUES " + "(" + name + "," + value + ")";
10    statement.execute(sql);

```

```

10    }
11
12
13 /**
14 * @description Permet de lire une partie ou toute la base de données
15 * @param select : '*' pour tout ou seulement une colonne => [
16   currDateTime], [registerName], [registerValue]
17 * @param options : peut être ORDER BY, WHERE, GROUP BY...
18 * @throws SQLException
19 */
20 public void select(String select, String options) throws SQLException{
21     String sql = "SELECT " + select + " FROM sensorValue " + options;
22     try {
23         rs = statement.executeQuery(sql);
24         while(rs.next()) {
25             System.out.println(rs.getTimestamp("currDateTime") + " " +
26                   rs.getString("registerName") + " " + rs.getFloat("registerValue"));
27         }
28     } catch(SQLException sq) {}
29 }
```

Listing 10 – Fonction Java des requêtes SQL

```

1 Adresse de l'automate (défaut : 192.168.1.4) :
2
3 Temps entre chaque lecture (défaut : 5000ms) :
4 10000
5 Nombre de répétitions de la lecture (défaut : 1) :
6 3
7 Adresse du registre à lire :
8 %MW150
9 Taille du registre à lire (défaut : 1) :
10 4
11 Données enregistrées :
12 192.168.1.4 | 1000 | 3 | 150 | 4
13 Si c'est OK, entrez 'ok'
14 ok
15 _____
16
17 Driver connecté !
18 Client connecté
19 === Contenu des registres ===
20 6400 0 10 38
21 12800 0 10 38
22 19200 0 10 38
23 === Contenu de la base de données ===
24 2021-08-19 15:09:49.700 MW150          6400.0
25 2021-08-19 15:09:59.920 MW150          12800.0
26 2021-08-19 14:10:10.147 MW150          19200.0
27 Driver déconnecté
28 Client déconnecté
```

Listing 11 – Exemple d'exécution sur un registre de type %MW avec enregistrement dans la base de données

Bibliographie

Bibliographie

- [1] (), Lab-STICC, adresse : <https://labsticc.fr/fr>. (Accédé : 30/08/2021).
- [2] DJI. (). “Matrice 100 - DJI,” adresse : <https://www.dji.com/fr/matrice100>. (Accédé : 30/08/2021).
- [3] ——, (). “Onboard-SDK - GitHub,” adresse : <https://github.com/dji-sdk/Onboard-SDK>. (Accédé : 30/08/2021).
- [4] WIKIPÉDIA. (). “UART,” adresse : <https://fr.wikipedia.org/wiki/UART>. (Accédé : 30/08/2021).
- [5] ——, (). “Baud (mesure),” adresse : [https://fr.wikipedia.org/wiki/Baud_\(mesure\)](https://fr.wikipedia.org/wiki/Baud_(mesure)). (Accédé : 30/08/2021).
- [6] DECAWAVE. (). “MDEK1001 Development Kit,” adresse : <https://www.decawave.com/product/mdek1001-deployment-kit/>. (Accédé : 30/08/2021).
- [7] ——, (). “Baud (mesure),” adresse : https://www.decawave.com/wp-content/uploads/2019/03/DRTLS_Manager_R2.apk. (Accédé : 30/08/2021).
- [8] S. O’HARA. (). “Enhanced Modbus library implemented in the Java programming language,” adresse : <https://github.com/steveohara/j2mod>. (Accédé : 30/08/2021).
- [9] ——, (). “Enhanced Modbus library implemented in the Java programming language,” adresse : <https://github.com/steveohara/j2mod/wiki>. (Accédé : 30/08/2021).
- [10] MICROSOFT. (). “Driver JDBC for SQL Server,” adresse : <https://docs.microsoft.com/fr-fr/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver15>. (Accédé : 30/08/2021).
- [11] DJI. (). “Matrice 100 - DJI,” adresse : <https://www.dji.com/fr/matrice100>. (Accédé : 30/08/2021).
- [12] ——, (). “Manifold - DJI,” adresse : <https://www.dji.com/fr/manifold>. (Accédé : 30/08/2021).
- [13] E. S. AGENCY. (). “Bancroft Method - Navipedia,” adresse : https://gssc.esa.int/navipedia/index.php/Bancroft_Method. (Accédé : 30/08/2021).
- [14] SPACE et G. L. / U. of TEXAS AT AUSTIN. (). “GPSTk - GitHub,” adresse : <https://github.com/SGL-UT/GPSTk>. (Accédé : 30/08/2021).

- [15] G. HUNTER. (). “Linux Serial Ports Using C/C++,” adresse : <https://blog.mbedded.ninja/programming/operating-systems/linux/linux-serial-ports-using-c-cpp/>. (Accédé : 30/08/2021).
- [16] ULTRALYTICS. (). “YoloV5 - GitHub,” adresse : <https://github.com/ultralytics/yolov5>. (Accédé : 31/08/2021).
- [17] TZUTALIN. (). “LabelImg. Git code (2015),” adresse : <https://github.com/tzutalin/labelImg>. (Accédé : 30/08/2021).