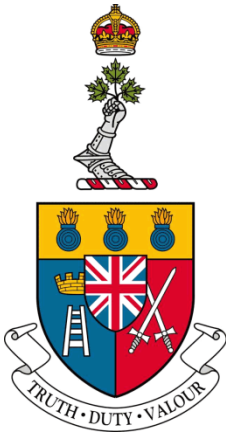
COLLÈGE MILITAIRE ROYAL DU CANADA
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET ÉLECTRIQUE



Document de conception détaillée (GEF455 - DID-07)

pour

Système de repérage aéroporté

Élof Mathieu Gagnon
Élof Olivier Sirois

Projet 455/15/10

Superviseur : Capt Vincent Roberge
Gestionnaire de projet : Capt Vincent Roberge
3 mars 2016

Table des matières

Liste des figures et tableaux

1. Introduction

1.1 Aperçu du projet

Notre projet de système de repérage aéroporté s'inscrit dans un contexte où l'usage de drones est de plus en plus courant dans l'exécution de différentes missions militaires et commerciales [1]. Un exemple typique de ce type d'utilisation est la recherche d'une embarcation de sauvetage en mer. Lors d'une telle mission de recherche de précision, un drone volant à une altitude relative de 150 mètres[2] pourrait recueillir des images servant à identifier une cible.

Les images et autres données captées par l'équipement de repérage installé sur le drone sont soit emmagasinées dans une mémoire sur l'appareil, ce qui retarde leur traitement, soit retransmises en direct à l'opérateur, ce qui peut être très lent compte tenu des limitations de bande passante. Dans le cas d'une analyse instantanée, une communication constante et de bonne qualité entre le drone et l'opérateur est nécessaire à l'accomplissement de ces missions, limitant ainsi la capacité de recherche des équipements actuels aux endroits où il est possible d'établir un lien de communication fiable. Dans une zone ennemie, il existe aussi un risque que l'ennemi capte ou brouille les signaux de communication, ajoutant de ce fait une limitation supplémentaire aux systèmes existants. Finalement, la vitesse d'analyse des données recueillies est aussi limitée par la capacité de ou des opérateurs à étudier une grande quantité de photos selon la surface couverte par le drone. Notre système de repérage aéroporté a pour objectif de mitiger ces problématiques.

1.2 Objectifs du projet

Le but de notre projet est de créer un système de repérage aéroporté pouvant être installé sur un drone afin de permettre l'automatisation de l'analyse de cibles survolées et du calcul de leurs positions en temps réel. Ce système doit analyser, sans intervention humaine, le terrain survolé afin de détecter la présence d'une cible, calculer la position exacte de la cible en considérant du relief du terrain et, au terme de la mission de recherche, communiquer à l'opérateur les informations pertinentes sur l'emplacement de la cible et une image de celle-ci. En fonctionnant de façon autonome, ce système devait permettre à un drone d'atteindre des surfaces de recherche qui ne seraient pas accessibles si un lien de communication permanent devait être maintenu en tout temps, soit par l'impossibilité d'établir lien, par une bande passante limitée ou encore par le risque que le signal soit capté ou altéré par l'ennemi. L'information devait être traitée par le module embarqué afin de ne pas avoir à transmettre de renseignements sensibles via les ondes radio, améliorant ainsi grandement la portée et la sécurité des opérations en permettant d'atteindre des zones où la communication ne serait pas possible et en évitant que les transmissions soient captées ou brouillées. De plus, le traitement de l'information par le dispositif devait assurer une analyse uniforme, rapide et précise des données photographiques recueillies, avec pour objectif de limiter les erreurs d'interprétations humaines tout en accélérant le déroulement de l'opération de repérage.

Dans le cadre de notre projet, nous avons utilisé un miniordinateur Raspberry Pi, auquel nous avons joint une caméra. Nous avons monté le tout sur une maquette afin de simuler un drone. Nous avons ensuite développé un logiciel pour nous permettre d'atteindre nos objectifs, soit le calcul de la position d'une cible et une interface de communication avec

l'utilisateur afin qu'il puisse interpréter les résultats retournés par le Raspberry Pi. Comme nous travaillons à l'intérieur, nous avons utilisé le système OptiTrack du laboratoire de robotique afin de simuler les coordonnées de localisation que nous aurions normalement pu obtenir du GPS et les angles d'inclinaisons de notre appareil que nous aurions normalement pu obtenir des instruments de vol de l'appareil.

1.3 Portée

Dans le cadre du cours, le projet fut effectué à une échèle réduite, mais réaliste, d'un facteur de 150 :1 afin que notre concept soit présentable à l'intérieur du laboratoire de robotique. Nous avons choisi une échelle de 150 :1 afin de refléter la réalité d'un drone qui volerait à une hauteur de 150 à 300 mètres, ce qui se traduit par une hauteur d'environ un à deux mètres dans l'arène. Notre projet sera démontré en assumant que le drone sur lequel notre équipement serait installé dispose d'un système d'autopilotage et est chargé d'un itinéraire préprogrammé. De plus, nous utilisons le système de localisation OptiTrack installé dans le laboratoire de robotique afin d'obtenir les coordonnées de localisation qui auraient normalement été fournies par le GPS et la plateforme inertielle du drone. Aussi, le projet fut réalisé à l'intérieur du cadre budgétaire restreint, une caméra moins performante qui n'est pas en mesure de prendre des relevés infrarouges a été utilisée. L'arène du laboratoire de robotique ne permet pas de recréer un environnement extérieur réel, toutefois un faux relief y est simulé et une carte topographique correspondant à celui-ci est chargée dans la mémoire du système de repérage. Finalement, afin de limiter la complexité et le risque de destruction de notre

projet, la démonstration ne sera pas effectuée par un vrai drone, mais plutôt à l'aide une maquette qui sera déplacée manuellement pour simuler un drone en vol.

Dans le cadre de la réalisation de notre projet, nous avons acquis la plupart des pièces matérielles, c'est-à-dire le Raspberry Pi, la caméra, la pile et les antennes de communications. Pour supporter notre équipement, nous avons utilisé un modèle de drone que nous avons trouvé sur internet et que nous avons pu bâtir à l'aide d'une imprimante 3D. Nous avons aussi développé un convertisseur de voltage. Finalement, nous avons développé le logiciel nécessaire à notre projet, cependant, nous faisons appel à plusieurs librairies préalablement développées et rendues disponibles sur internet.

1.4 Aperçu du document

Ce document de conception détaillé a pour objectif de présenter et de décrire l'intégralité du système que nous avons développé afin de répondre à la problématique présentée dans le préambule. Nous y présenterons le détail des besoins que notre projet tente de satisfaire, sa conception fonctionnelle, les détails de conception, la description de l'équipement, les résultats obtenus, ainsi que le résumé de nos réalisations accompagnées d'une discussion sur ce qui pourrait être amélioré pour rendre notre projet plus performant.

2. Document de références

La première série de documents qui suivent porte sur des exemples d'utilisation de drone pour diverses missions et sur lesquelles nous nous sommes basés afin de développer notre concept. Les autres documents présentent le cadre de référence dans lequel le projet a été

conçu, ainsi que notre énoncé initial des besoins et notre spécification préliminaire de conception.

[1] J. ATTARIWALA, “Report on UVS technology: Canadian companies at cutting edge”, Canadian Defence Review, vol. 19, Automne 2015.

Cet article mentionne l’importance grandissante de l’utilisation de drone dans des applications diverses, tant dans le domaine militaire que commercial.

[2] H. WILLIAMS, “US Army to enhance Puma UAS fleet”, HIS Jane’s International Defence Review, vol. 48, Octobre 2015.

Cet article présente un exemple d’utilisation de petit drone dans le cas de mission militaire de recherche et de livraison d’objet à une cible précise. Il spécifie entre autres l’altitude moyenne de vol pour ce type de drone.

[3] Département de génie électrique et informatique, “Énoncé de travail”, Collège militaire royal du Canada, Septembre 2015.

Ce document précise la nature du travail attendu des étudiants du département pour le projet final de génie électrique et informatique.

[4] O. SIROIS & M. GAGNON, “Énoncé des besoins”, Collège militaire royal du Canada, Octobre 2015.

Ce document définit les fonctions que doit accomplir notre design et les paramètres dans lesquels il doit atteindre ses objectifs.

[4] O. SIROIS & M. GAGNON, “Spécification préliminaire de conception”, Collège militaire royal du Canada, Novembre 2015.
Ce document définit la conception préliminaire du design et présente le calendrier initial de réalisation du projet.

[5] "Vision – Student Robotics", *Studentrobotics.org*, 2016. [Online]. Available: https://www.studentrobotics.org/trac/wiki/Vision. [Accessed: 16- Mar- 2016].

[6] "File:HSV color solid cylinder alpha lowgamma.png - Wikimedia Commons",*Commons.wikimedia.org*, 2010. [Online]. Available: https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder_alpha_lowgamma.png. [Accessed: 16- Mar- 2016].

3. Spécification des besoins

Cette section présente les exigences minimales du projet que nous nous proposions d’atteindre, divisées parmi les catégories suivantes : exigences fonctionnelles, exigences d’interface, exigences de performance et de synchronisation, de simulation et diverses, ainsi que les restrictions en terme de mise en œuvre et d’échéancier.

3.1 Exigences fonctionnelles

La liste suivante énumère les principales exigences fonctionnelles que nous avions incluses dans notre énoncé des besoins.

FR-01 : Prise de Photo – Le système de repérage doit être en mesure de photographier les

cibles.

FR-02 : Identification d'une cible – Le système de repérage doit être en mesure d'analyser les informations recueillies afin d'identifier une cible déterminée par l'opérateur.

FR-03 : Calcul de la localisation d'une cible – Le système doit être en mesure de calculer la localisation sur les axes x, y, z de l'arène d'une cible au sol selon les exigences de résolution précisées dans les exigences de performance.

FR-04 : Communication des résultats – Le système doit être en mesure de transmettre à l'opérateur la localisation des cibles identifiées.

FR-05 : Alimentation sécurisée – Le système d'alimentation doit fournir une tension de 5 volts, avec un écart maximal de 10% (spécifications du Raspberry Pi), de plus, il doit être doté d'une protection contre les pointes de tension et prévenir l'utilisateur en cas de basse tension.

FR-06 : Sauvegarde des informations – Le système doit être en mesure de garder les photos et les coordonnées des cibles repérées en mémoire.

FR-07 : Relief du terrain– Le système doit tenir en compte du relief du terrain dans le calcul de la position de la cible à l'aide d'une carte topographique préenregistrée.

FR-08 : Indépendance du système – Le système doit être capable d'opérer de manière autonome une fois la mission lancée.

3.2 Exigences d'interface

La liste qui suit énumère les différentes exigences d'interface que nous avions incluses dans notre énoncé des besoins.

IR-01 : Communication avec le système OptiTrack – notre système devra être en mesure

de recevoir les informations de localisation transmises par le réseau OptiTrack.

3.3 Exigences de performance et de synchronisation

La liste qui suit énumère les différentes exigences de performance et de synchronisation que nous avions incluses dans notre énoncé des besoins.

PR-01 : Fréquence d'analyse d'image – Le système doit être en mesure d'analyser une nouvelle image à toutes les secondes.

PR-02 : Durée de mission – Le système devra être en mesure d'opérer pendant un minimum de 4 heures avec une source d'énergie autonome.

PR-03 : Champs de vision - Le système de repérage devra être capable de détecter une cible terrestre lorsque celle-ci se retrouve entièrement dans son champ de vision.

PR-04 : Taille minimale de la cible - Le système de repérage devra être en mesure de détecter n'importe qu'elle cible circulaire qui a plus de 45cm² de surface continue.

PR-05 : Résolution de repérage – Le système de repérage devra être en mesure de repérer au moins 5 cibles différentes avec une distance minimale de 10 cm entre les cibles et de conserver une image de chacune de ces cibles.

PR-07 : Sécurisation du système : Les communications avec l'opérateur du système doivent être encryptées.

4. Conception fonctionnelle

Cette section présente une vue générale du fonctionnement de notre prototype. La description en détail de chacun des modules, des interactions entre eux, ainsi qu'avec les utilisateurs du système et son environnement seront présentés dans la prochaine section. La figure 4-1 présente l'architecture de notre prototype.

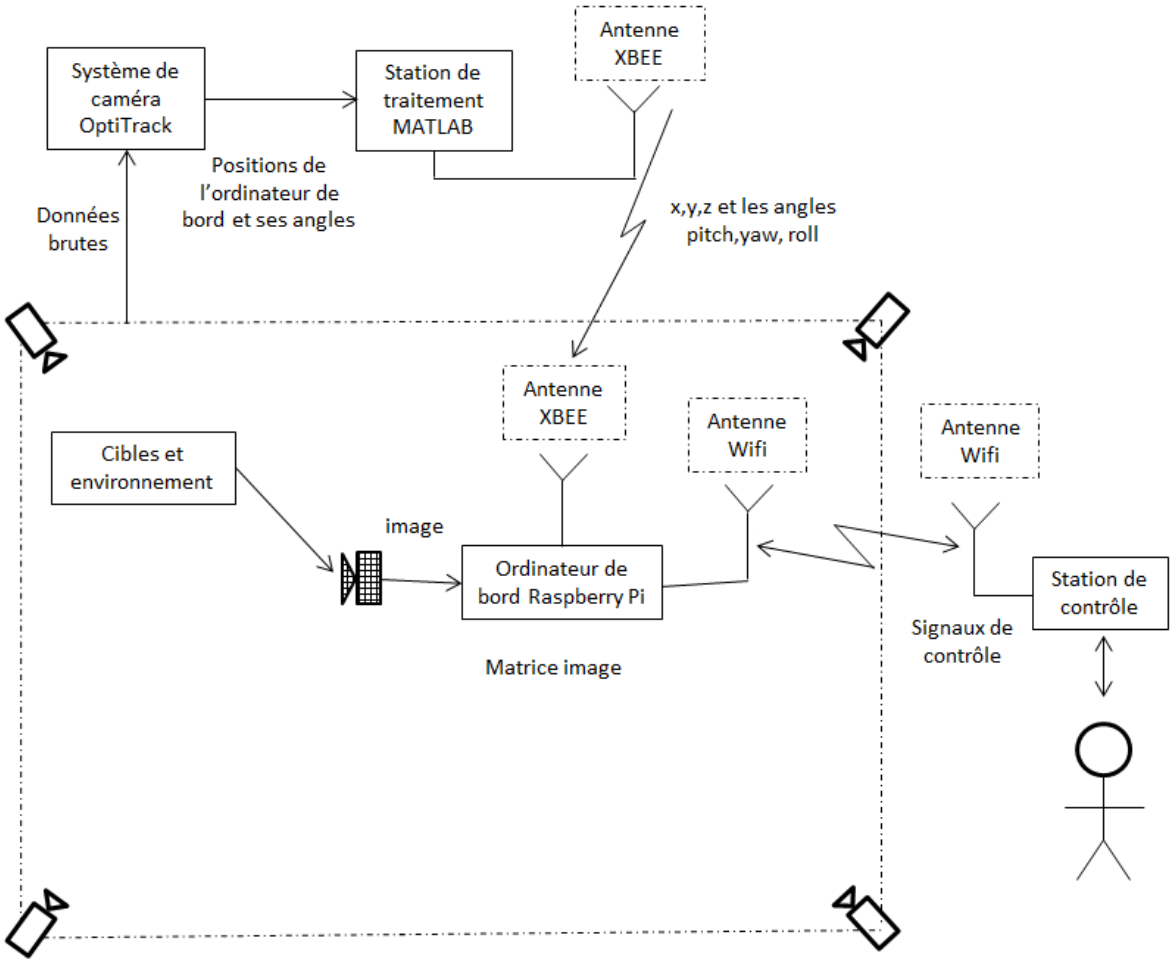


Figure 4-1 : Architecture du produit

Afin de faciliter la description du fonctionnement de notre système de repérage, nous l'avons divisé en quatre modules principaux. Premièrement, la partie la plus centrale est le module que nous installons sur notre drone, il comprend un ordinateur Raspberry Pi relié à une caméra et alimenté pas un combo bloc-pile et un convertisseur CC. Le deuxième module comprend un ordinateur sur lequel est installé Matlab. Cette station nous permet d'interroger le système OptiTrack, notre troisième composante, pour connaître la position et l'inclinaison de notre drone à l'intérieur de l'arène et de diffuser cette information par ondes radios. Ces deux modules nous permettent d'émuler un

système GPS à l'intérieur du laboratoire. Finalement, une station de contrôle permet à l'opérateur de communiquer avec le Raspberry Pi afin de lancer une mission ou de prendre connaissance des résultats de celle-ci.

5. Détails de conception

Cette section approfondit les notions présentées à la section précédente. Le fonctionnement de chacun des quatre modules présentés sera analysé de même que la façon dont il communique avec les autres modules, les usagers et son entourage.

5.1 Aperçu général

Comme nous l'avons illustré à la figure 4-1, notre système comprend quatre modules soit le système de caméra OptiTrack, la station de traitement MATLAB, la station de contrôle de l'usager et l'ordinateur de bord Raspberry Pi. Ces modules interagissent entre eux de façon unidirectionnelle ou bidirectionnelle selon le type de données échangées.

Nous utilisons l'arène OptiTrack du laboratoire de robotique pour le développement de notre concept puisqu'elle nous permet d'obtenir en temps réel la localisation et l'inclinaison exactes de notre drone. Il est essentiel d'obtenir ces données au moment de la prise de photo afin de calculer un vecteur à projeter qui nous permettra de déterminer de la localisation d'une cible préalablement identifiée. Nous utilisons une station Matlab sur lequel s'exécute en boucle un script qui interroge le server OptiTrack pour obtenir les données de localisation de notre drone pour ensuite les encapsuler dans un format intelligible pour le programme du Raspberry Pi. La station Matlab est aussi responsable de diffuser ces informations à l'aide d'une antenne XBee.

Le module principal de notre projet est formé d'un ordinateur Raspberry Pi, alimenté par

un bloc pile, d'une caméra afin d'analyser le terrain et de rechercher une cible, d'une antenne XBee afin de recevoir les coordonnées de localisation du système OptiTrack et d'une antenne Wifi afin que l'opérateur puisse communiquer avec le Raspberry Pi à partir de la station de contrôle. Cette dernière n'est qu'un ordinateur équipé lui aussi d'une antenne Wifi et à partir duquel l'opérateur peut établir soit une communication SSH ou VNC afin de commander le Raspberry Pi ou d'obtenir les résultats d'une mission.

5.2 Limites

Notre projet peut facilement être adapté pour être utilisable dans un autre environnement. Puisque nous avons développé notre concept dans le but d'effectuer une démonstration à l'intérieur du laboratoire, nous utilisons le système OptiTrack afin d'obtenir la localisation de notre drone et son inclinaison. Si l'on veut utiliser notre système à l'extérieur, il faudrait y ajouter un GPS afin d'obtenir la localisation de l'appareil. De plus, il faudrait obtenir l'inclinaison de l'appareil, possiblement en utilisant les informations provenant d'accéléromètres ou de gyroscopes installés sur le drone lui-même, l'ordinateur de vol de l'appareil pourrait possiblement aussi être en mesure de fournir ces informations. Pour une utilisation à une altitude plus élevée, une caméra dotée d'une meilleure résolution pourrait être nécessaire. De même, si l'on voulait effectuer de la détection de personnes, une caméra infrarouge pourrait serait nécessaire. Finalement, afin de communiquer avec le drone, un ordinateur portable ou, à la limite, un téléphone intelligent, serait nécessaire pour communiquer avec le drone pour lancer une mission ou obtenir les résultats de celle-ci.

5.3 Description des modules

Les quatre sous-sections qui suivent présentent le détail du fonctionnement et de la conception des quatre modules de notre système de repérage. Dans le cas du système OptiTrack et de la station de contrôle, le détail du fonctionnement est présenté, par contre, comme ces équipements sont utilisés « tel quel », nous ne fournissons pas de détails de conception. Dans le cas de la station Matlab et encore plus dans le cas du Raspberry Pi, en plus des détails de fonctionnement, une revue complète de la conception sera présentée.

5.3.1 Système de caméra OptiTrack

Le système de caméra OptiTrack est un équipement utilisé afin de suivre les mouvements d'un objet à l'intérieur d'une arène. Il fonctionne à l'aide de caméras qui recherchent des billes d'identification installées sur l'objet à suivre. Nous utilisons OptiTrack afin d'obtenir la localisation exacte de notre drone à l'intérieur de l'arène selon un système de référence cartésien. En plus de fournir les coordonnées de localisation de notre drone, il nous permet aussi d'obtenir les angles en roulis, tangage et lacet de notre drone. Ces informations sont emmagasinées sur un serveur à partir duquel nous pouvons formuler des demandes (*requests*) afin d'obtenir la localisation de notre appareil selon ses coordonnées en x, y, z et ses angles d'inclinaisons relatifs en quaternions. Nous n'avons aucun contrôle sur son fonctionnement interne.

5.3.1.1 Système de caméra OptiTrack – spécifications et contraintes

Le système OptiTrack étant un système fermé, nous devons utiliser les outils fournis par le concepteur du système afin d'obtenir les données de localisation. Ces outils ne permettent pas de diffuser directement les coordonnées de l'objet à suivre dans l'arène à

l'aide d'une antenne XBee. Nous avons donc choisi de communiquer avec le serveur OptiTrack en utilisant Matlab via un réseau câblé existant.

5.3.1.2 Système de caméra OptiTrack – conception

Aucune conception ne fût réalisée du côté du système OptiTrack, celui-ci répond aux demandes effectuées par notre station Matlab dans le cadre normal de son fonctionnement. La seule opération que nous devons effectuer avec OptiTrack est une calibration périodique du système.

5.3.2 Station de traitement MATLAB

La station Matlab est un outil que nous avons développé afin de pallier à l'impossibilité de diffuser directement les coordonnées du drone à partir du système OptiTrack. Cette station consiste en un ordinateur, relié au réseau câblé d'OptiTrack sur lequel le logiciel Matlab est installé. Nous avons développé un script Matlab qui interroge périodiquement le serveur OptiTrack pour obtenir les données de localisation du drone selon les axes x, y et z ainsi que les angles relatifs de positionnement du drone en quaternions. Ces données sont ensuite formatées et encapsulées dans une chaîne de caractère que nous diffusons continuellement par ondes radios à l'antenne d'une antenne XBee.

5.3.2.1 Station de traitement MATLAB – spécifications et contraintes

Notre station Matlab fonctionne sous Microsoft Windows 7 et nous utilisons la version 2014 du logiciel Matlab. Nous avons choisi d'utiliser une antenne XBee, connectée à un port USB de notre station, puisqu'elle est très simple à configurer, flexible et compatible

avec les différents systèmes d'exploitation que nous utilisons. La portée maximale du type d'antenne XBee que nous utilisons est de 30 mètres à l'intérieur d'un édifice, ce qui est suffisant pour couvrir le laboratoire au complet.

5.3.2.2 Station de traitement MATLAB – conception

L'essentiel de la conception de ce module réside dans le code qui permet d'interroger le serveur OptiTrack, de traiter l'information et de la relayer à l'antenne XBee. Le flot internet des données est expliqué à la section 5.3.2.2.1. La base de ce code est issue d'un exemple fourni par OptiTrack et nous l'avons adapté à nos besoins. Les éléments principaux ainsi que les changements que nous avons effectués au code fourni par OptiTrack sont également présentés à la section 5.3.2.2.2. Le code entier du script se retrouve à l'annexe A, certaines fonctions du code ne sont pas utilisées, mais nous avons préféré les laisser en place afin de faciliter de futurs développements

5.3.2.2.1 Flot des données

Le flot des données est présenté à la figure 5-1. Le traitement des données débute lorsque Matlab reçoit les données brutes de localisation du serveur OptiTrack. Ces données comprennent trois valeurs de positions : x, y et z, ainsi que quatre valeurs d'inclinaisons, en quaternions : qx, qy, qz et qw. Ensuite, à l'aide d'une fonction incluse dans la suite d'outils NatNet d'OptiTrack, nous effectuons la conversion des angles en quaternions pour obtenir trois inclinaisons : le roulis, le tangage et le lacet. Ces unités sont plus faciles à utiliser dans le cadre de notre projet. Puis, nous encodons dans une chaîne de caractère débutant par une étiquette DEBUT, suivi d'un numéro de séquence et se

terminant par une autre étiquette FIN, les 3 données de positions et les 3 valeurs d'inclinaison. Finalement, la chaîne de caractère est transmise au Raspberry Pi en utilisant l'antenne XBee. Les étiquettes et le numéro de séquence sont nécessaires afin que le Raspberry Pi puisse rejeter les transmissions reçues partiellement ou en double.

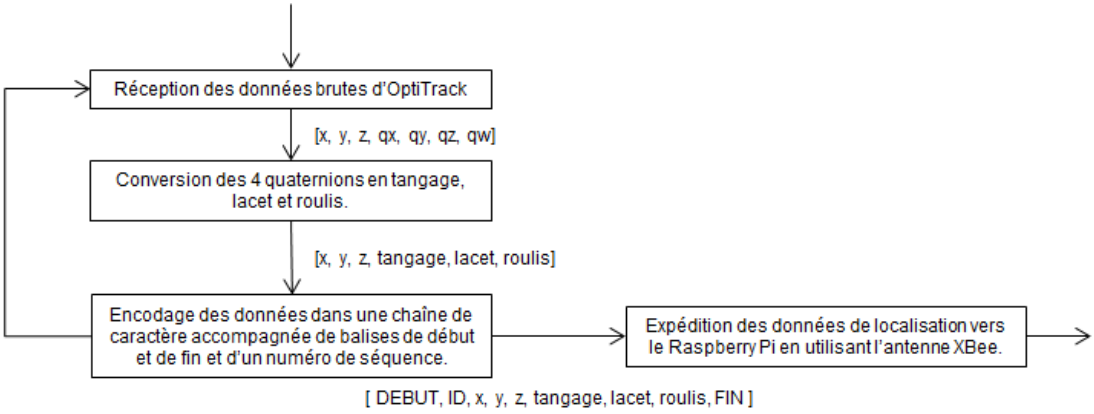


Figure 5-1 : Flot des données du script Matlab

5.3.2.2.2 Description du fonctionnement

La figure 5-2 présente les trois séquences principales de notre script Matlab. La première section concerne l'initialisation du port pour l'antenne XBee, nous utilisons les paramètres recommandés par le fabricant. Dans la deuxième séquence de la figure 5-2, nous préparons les données brutes reçues d'OptiTrack, dans un premier temps pour convertir les quaternions, puis pour pouvoir les transmettre à l'aide de l'antenne XBee. Les données de position x, y et z sont temporairement stockées dans des variables séparées, les positions seront multipliées par 1000 avant d'être transmises pour obtenir des millimètres. Les quatre quaternions sont ensuite extraits et placés dans une matrice. Cette matrice doit subir deux opérations pour être transformée en radian. Finalement, nous stockons convertissons ces angles en degrés avant de les stocker dans des variables

individuelles. La troisième séquence illustre la façon dont sont transmises les données par l'antenne XBee. Il est à noter qu'à des fins de contrôle à la réception, nous avons ajouté des balises DEBUT et FIN, ainsi qu'un numéro de séquence.

```
1.  %Initialisation du port pour le XBee
2.  XBEE = serial('COM3','BaudRate',115200, 'DataBits', 8, 'Parity',
    'none', 'StopBits', 1, 'FlowControl', 'none')
3.  fopen(XBEE)


4.  %Preparation des donnees a envoyer
5.  POSx = data.RigidBodies(1).x;
6.  POSy = data.RigidBodies(1).y;
7.  POSz = data.RigidBodies(1).z;
8.  q = quaternion(data.RigidBodies(1).qy, data.RigidBodies(1).qx,
    data.RigidBodies(1).qz, data.RigidBodies(1).qw);
9.  qRot =   quaternion( 0, 0, 0, 1);
10. q = mtimes(q, qRot);
11. angles = EulerAngles(q,'zyx');
12. angleYaw = -angles(1) * 180.0 / pi;
13. angleRoll = angles(2) * 180.0 / pi;
14. anglePitch = -angles(3) * 180.0 / pi;

15. %Envoie des donnees
16. datatosend = [123.456, 1000*POSx, 1000*POSy, 1000*POSz, an-
    glePitch,    angleYaw, angleRoll, idx]
17. for i = 1:length(datatosend)
18.         datatosend_conv{i} = num2str(datatosend(i), '%18.8f');
19.         LengthString = LengthString + length(num2str(datatosend(i),
                '%18.8f'));
20. end
21. fwrite(XBEE, 'DEBUT')
22. fwrite(XBEE, ' ')
23. for j = 1:length(datatosend_conv)
24.         fwrite(XBEE, datatosend_conv{j})
25.         fwrite(XBEE, ' ')
26. end
27. fwrite(XBEE, 'FIN')
28. fwrite(XBEE, ' ')
29. fwrite(XBEE, hex2dec('0A'), 'uint8')
```

Figure 5-2 : 3 séquences du script Matlab

5.3.2.2.3 Traitement des erreurs

Il n'y pas de contrôle d'erreur qui est faite à ce stade. La vérification des données est faite par le Raspberry Pi lorsqu'il reçoit les informations transmises par le XBee.

5.3.3 Ordinateur de bord Raspberry Pi

L'ordinateur de bord Raspberry Pi est l'élément central de notre projet. Il s'agit d'un

petit ordinateur embarqué que nous pouvons installer sur notre drone et qui est doté d'une alimentation autonome, de deux antennes et d'une caméra. Le Raspberry Pi est responsable d'analyser les images captées par sa caméra afin d'identifier une cible, de calculer la localisation exacte de cette cible en fonction d'une carte d'élévation préalablement chargée dans la mémoire et de retourner les résultats à l'opérateur. La section qui suit décrit son fonctionnement en détail.

5.3.3.1 Ordinateur de bord Raspberry Pi – conception, spécifications et contraintes

Dans le but de démontrer notre concept, nous avons construit une maquette sur laquelle nous avons installé le Raspberry Pi. Il s'agit d'un modèle de quadcoptère que nous avons trouvé sur internet et que nous avons imprimé en 3D. Nous avons ensuite fixé le boitier du Raspberry Pi sous la maquette, de façon à ce que la caméra qui est installée à l'intérieur du boitier pointe vers le sol. Sur la maquette, nous avons également installé la pile qui fournit l'énergie nécessaire au fonctionnement du Raspberry Pi. La tension fournie par ce bloc pile étant plus élevée que les 5 volts requis par le Raspberry Pi, nous avons conçu un convertisseur que nous avons également installé sur notre maquette. Deux antennes sont reliées aux ports USB du Raspberry PI, une antenne XBee afin de recevoir les coordonnées de localisation de la station Matlab et une antenne Wifi afin que l'opérateur puisse communiquer avec ce dernier à partir de la station de contrôle. Le programme d'identification de cible est chargé sur une carte microSD, celle-ci agit comme un disque dur pour le Raspberry Pi. Il existe peu de contraintes pour l'opération du module, il doit cependant se trouver à l'intérieur de l'arène OptiTrack afin que sa position soit détectée.

5.3.3.2 Ordinateur de bord Raspberry Pi – théorie du calcul du positionnement

Pour calculer la position d'une cible, nous avons besoin de trois données : une image de la cible identifiée, la position du drone ainsi que son orientation à l'endroit où a été prise l'image et une carte d'élévation représentant le relief de l'endroit où a été prise la photo. La prise de l'image est la partie la plus importante du calcul de l'emplacement de la cible. Cette opération doit être effectuée immédiatement après avoir obtenu la position du drone pour nous assurer que les données de localisation reçues soient les plus fidèles possible. Après avoir pris l'image, nous devons être en mesure de pouvoir identifier une cible sur celle-ci. Dans l'étendue de notre projet, nous avions spécifié que nous recherchions un objet de couleur rouge (choix arbitraire). Nous y parvenons en changeant le format de notre image de RGB à HSV, puis nous filtrons les couleurs non désirées, finalement nous appliquons un détecteur de blob. Cette détection de blob est réalisée à l'aide de la librairie OpenCV. Nous avons choisi d'utiliser cette libraire parce qu'elle est très bien documentée et qu'elle est gratuite. Celle-ci permet entre autres d'analyser très rapidement l'image filtrée et d'identifier les différentes agglomérations d'objets rouges dans l'image, c'est-à-dire les cibles.

L'étape suivante dans l'analyse d'image est l'obtention d'un vecteur à partir de ces cibles. Pour y parvenir, nous avons besoin des paramètres intrinsèques de la caméra. Ces paramètres sont faciles à obtenir à l'aide de l'outil calibration Bouguet. L'outil a simplement besoin d'analyser certaines images spécifiques prises par la caméra et peut déduire ces paramètres intrinsèques. Les résultats obtenus de notre calibration sont présentés à la figure 5.3.

Focal Length :	fc =	634.78213 , 636.25833		+/-	11.76136 , 11.43559	
Principal point :	cc =	320.88147 , 240.87371		+/-	1.91534 , 4.78356	
Skew :	alpha_cc =	0	+/-	0		
Distortion :	kc =	0.09074 , -0.30011	, 0.00074	, -0.00030	, 0.00000	
Pixel Error :	err =	0.34624 , 0.23339				

Figure 5.3 : Données de calibrations

Nous retrouvons à la figure 5.3 les données qui sont essentielles pour la calibration de la caméra. Dans le tableau, *fc*, désigne les longueurs focales en X et en Y respectivement. La prochaine valeur, *cc*, représente le point principal de la caméra, c'est-à-dire le pixel où se trouve le centre de l'image. La valeur suivante, *alpha_cc*, représente la rotation de l'écran par rapport à son centre. La matrice *kc* comprend les différents coefficients de distorsions de la caméra. Finalement, la valeur *err* représente l'erreur en nombre de pixels par rapport à la position réelle de l'objet analysé.

Après l'analyse des cibles, nous pouvons identifier avec précisions le centre de masses de chacune des différentes agglomérations trouvées. La librairie OpenCV possède une fonction qui sert à convertir des couples de pixels en vecteur normalisé. Cette fonction a comme paramètres d'entrées les couples de pixels à redresser ainsi que les différents coefficients de distorsions propres à la caméra utilisée. La figure 5.4 représente les données obtenues de cette opération

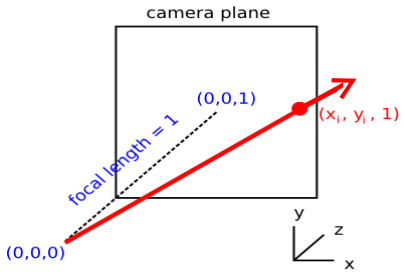


Figure 5.4 : Représentation graphique du vecteur normalisé (xi, yi, 1)

Après avoir obtenu les vecteurs normalisés pour chacune des cibles, nous devons les transposer dans le système de référence globale. Pour faire cela, nous devons transformer l'origine et tous les points des cibles en deux opérations. Une première transformation pour aller du système de référence de la caméra au système de référence du drone, puis une autre transformation pour passer du système de référence du drone au système de référence global (OptiTrack). Après avoir transformé les vecteurs, il nous restera simplement qu'à trouver l'intersection du vecteur sur la carte d'élévation pour déterminer l'emplacement de la cible.

La transformation des vecteurs est relativement simple. Nous devons nous servir d'une matrice de transformation T, représentée à l'équation 5.5, avec les paramètres requis pour changer de systèmes de référence.

$$T=\begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma)-\sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)cox(\gamma)+\sin(\alpha)\sin(\gamma) & x_t \\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma)+\cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma)-\cos(\alpha)\sin(\gamma) & y_t \\ -\sin(\beta) & \cos(\beta)\sin(\gamma) & \cos(\beta)\cos(\gamma) & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Équation 5.5 : Matrice de transformation T

Où α correspond à l'angle de tangage, β à l'angle de lacet, γ à l'angle de roulis, x_t à la translation en x, y_t à la translation en y et z_t à la translation en z.

Dans cette matrice, l'ordre d'opération est : roulis → lacet → tangage → translation. L'objectif de cette étape est d'effectuer les rotations et les translations nécessaires pour déplacer le vecteur origine de l'étape 3 et de le placer à l'emplacement exact de la caméra dans l'espace du système OptiTrack. Les seules informations que nous recevons du

système OptiTrack sont les coordonnées et les angles d'inclinaison du centre de masse de notre drone. Cependant, comme la position et l'orientation de la caméra ne correspondent pas nécessairement au centre de masse du drone, nous devons donc effectuer deux grandes transformations pour placer le vecteur origine au bon emplacement. La première transformation consiste à mettre la position de la caméra par rapport au centre de masse du drone et la deuxième permet de déplacer le vecteur origine à l'emplacement du drone. La figure 5.6 présente un croquis qui illustre ces deux transformations, accompagnée de quelques explications.

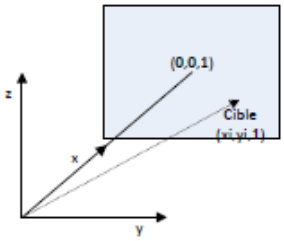


Image dans le plan de référence de la caméra, la cible se trouve à la position xi,yi,1 trouvé à l'aide de l'équation 4-4

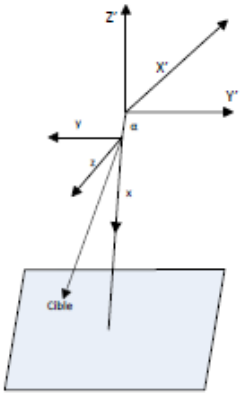


Image dans le plan de référence du drone. On peut calculer la position de la cible à l'aide de l'équation ci-dessous :

$$Cible = T_{drone \rightarrow caméra} * \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

La matrice T est composé des trois rotations d'axes (x', y', z')→ (x,y,z) et des translations α

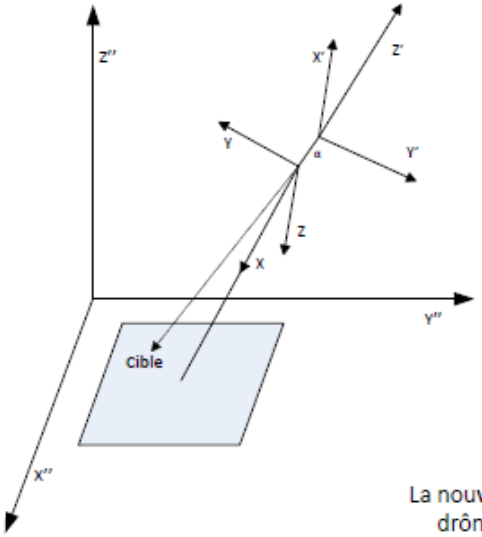


Image dans le plan de référence global. On peut calculer le vecteur de la cible à l'aide de l'équation ci-dessous :

$$Cible = T_{global \rightarrow drone} * T_{drone \rightarrow caméra} * \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

La nouvelle matrice est composé des trois rotations d'axe du drône dans l'arène OptiTrack avec sa position physique comme étant les trois translation dans le plan.

Figure 5.6 : Détails des transformations du système de référence de la caméra au système global

Maintenant que nous avons un vecteur transformé dans le système de référence global, nous pouvons le projeter sur notre carte d'élévation. Cette étape est la plus simple des trois. Nous allons premièrement calculer le vecteur allant du drone jusqu'aux cibles. Puis, nous allons prendre la position du drone dans le système OptiTrack et y ajouter le vecteur jusqu'à ce qu'il fasse intersection avec la carte d'élévation. Notre carte d'élévation, préalablement placée en mémoire, consiste en une matrice 2X2. Lors que le vecteur la rencontre, nous pouvons extraire la position en X et en Y. Il ne suffit que de la retransmettre à la station de contrôle. La figure 5.7 illustre ce que nous venons de décrire.

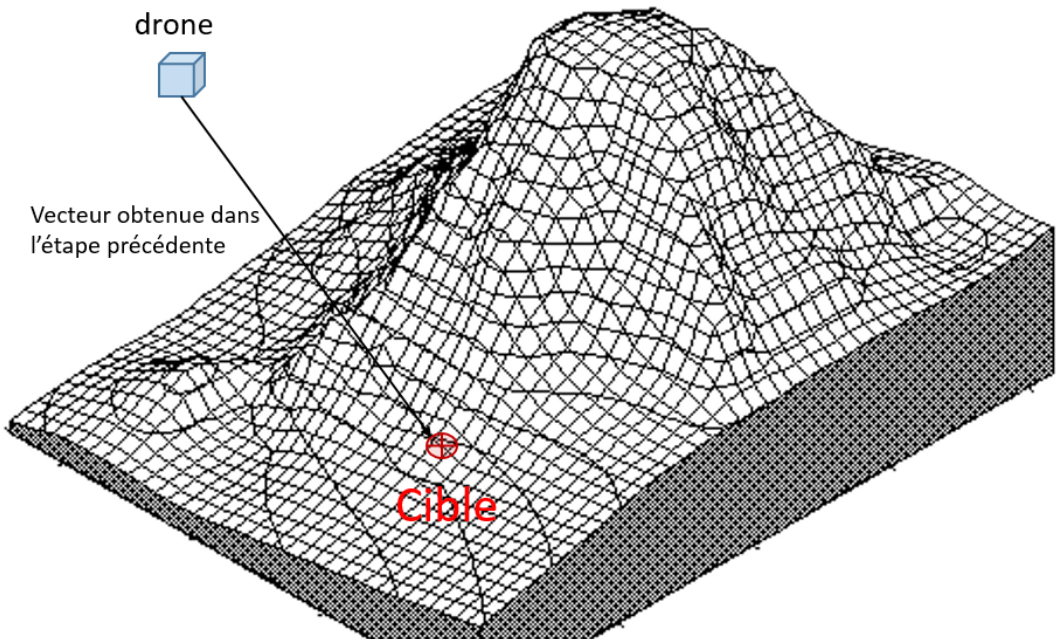


Figure 5.7 : Intersection sur la carte d'élévation

5.3.3.3 Ordinateur de bord Raspberry Pi – analyse du code informatique

Voici maintenant une analyse détaillée du code informatique que nous avons conçu pour identifier les cibles et calculer leurs localisations. Pour notre projet, nous avons utilisé

Python 2.7.9 comme langage de programmation. Nous avons choisi ce langage pour sa simplicité et l'impressionnante quantité de librairies disponibles étant donné l'envergure de notre projet et de nos contraintes de temps. Les modules que nous avons utilisés pour le projet sont les suivants :

a. CV2 : Un des modules centraux du projet, il permet le traitement des images ;

b. NP : Deuxième module central du projet, il exécute les opérations matricielles ;

c. CMATH : Module permettant les opérations cos et sin, essentiel pour les calculs des vecteurs ;

d. SciPy : Module permettant le calcul d'une matrice inverse, fonction non incluse dans le module NP ;

e. PySerial : Module permettant la communication avec l'antenne XBEE ;

f. OS : Module permettant d'appeler des commandes du système d'opération. Utilisé principalement pour le formatage de la sortie.

La figure 5-8 présente un diagramme de flot qui explique, à un haut niveau, le déroulement séquentiel de notre programme lors de son exécution. Les chiffres encerclés font référence aux différentes étapes qui seront analysées par la suite.

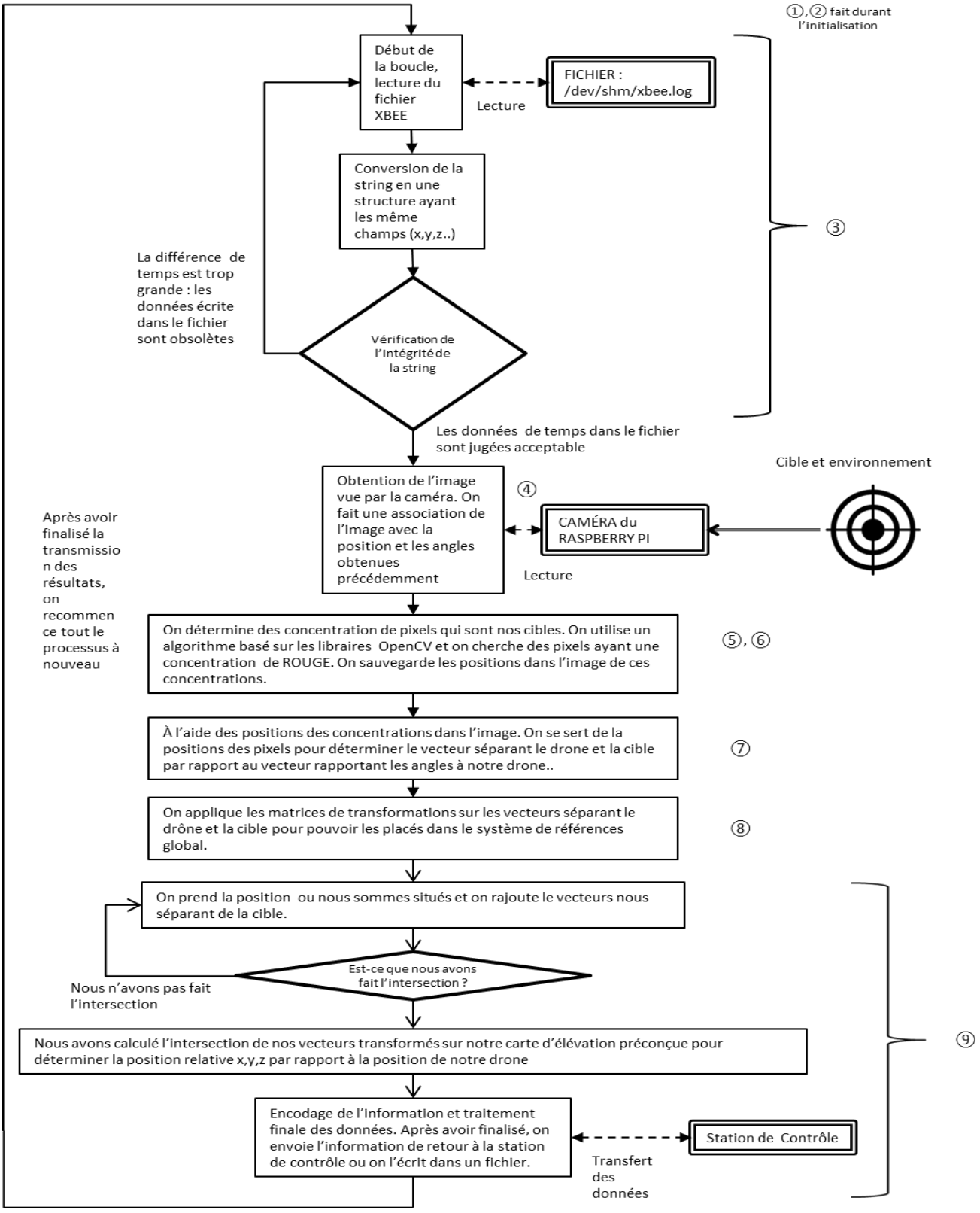


Figure 5.8 : Diagramme de flot du programme avec annotations

Notre programme commence premièrement par initialiser la carte d'élévation, une matrice 650 X 650. Une portion de ce code est présenté à la figure 5.9, le code complet comprend plusieurs doubles boucles *for* imbriquées afin de décrire l'élévation du terrain. Les valeurs dans la matrice représentent l'élévation en cm tandis que les indices représentent la position de la carte dans le système de référence.

```
1.      #initialisation de la matrice d'élévation, mesures en cm, 0 =
        plancher
2.
3.      Elevation = [ [ 0 for i in range(650) ] for j in range(650) ]
4.      XOFFSET = 325
5.      YOFFSET = 325
6.
7.      for i in range (0,250):
8.      for j in range (0,250):
9.             Elevation[i][j] = 0
10.
11.     for i in range (0,250):
12.     for j in range (250,400):
13.            Elevation[i][j] = 60
```

Figure 5.9 : Code d'initialisation de la carte d'élévation ①

La prochaine étape (2) comprend l'initialisation des variables globales. Nous débutons par l'initialisation de l'antenne XBEE comme étant un objet de type serial. Nous continuons avec les initialisations des différentes matrices de distorsion et de coefficients intrinsèques. Nous avons aussi ajouté dans notre projet une option débogage. Cette option que nous verrons plus en détail après nous permet de nous servir de notre projet sans nous servir du système OptiTrack. Ces détails de code informatique sont présentés à la figure 5.10.

```
1.      #initialisation of the global variables
2.      ser = serial.Serial("/dev/XBEE", 115200)
3.
4.      matrice_transfo_static = np.array([[1, 0, 0, 3.3],[0 , 1, 0,
        -1.0], [0,
5.      0,     1, -5.5], [0, 0, 0, 1]])
6.      K_Mat = np.array([[634.782, 0, 320.8814], [0, 636.25833,
        240.87371], [0,     0, 1]], dtype = np.float32)
7.      #K_inv = linalg.inv(K_Mat)
```

```
 8.      kc = np.array([0.0974, -0.3001, 0.00074, -0.00030, 0.0000], dtype
         =
 9.      np.float32)
10.
11.     #matrice_transfo = np.dot(test1, matrice_transfo_static
12.     #if you want to debug and test without optitrack
13.     OPTITRACK_OVERRIDE = 0
14.     XSCALING = 12.25
15.     YSCALING = 12.458
16.     ZSCALING = 12.5
```

Figure 5.10 : Code d'initialisation des variables et des matrices ②

Maintenant que nous avons initialisé nos variables globales, nous pouvons poursuivre avec le cœur de notre programme. La première partie est une boucle qui sert tout simplement à attendre une donnée acceptable dans le buffer de notre antenne XBee. Pour déterminer si la donnée reçue est bonne, nous débutons par séparer les éléments de la chaîne, puis nous vérifions si la chaîne nous est parvenue intacte. Lors de l'envoi de la chaîne de caractères de la station Matlab, nous incorporons toujours les identificateurs 'DEBUT' et 'FIN', placés respectivement au début et à la fin de la chaîne. Si ceux-ci ne se retrouvent pas dans la chaîne reçue, on recommence la boucle jusqu'à temps que cette condition soit remplie, tel que présenté à la figure 5.11.

```
 1.     print('Debut du programme')
 2.     while(1):
 3.     if OPTITRACK_OVERRIDE == 0:
 4.     #INSERT XBEE READING HERE
 5.            while (1) :
 6.                   try:
 7.                          # _, frame = cap.read()
 8.                          rawData = ser.readline()
 9.                          parsedData = rawData.split(" ")
10.                          if parsedData[0] == 'DEBUT':
11.                                 if len(parsedData) > 10 :
12.                                        if parsedData[10] == 'FIN':
13.                                               #print(parsedData)
14.                                               break
15.                                        else:
16.                                               print ('No good')
17.                                 else :
18.                                        print ('NO GOOD')
19.                          else :
20.                                 print('No GOOD')
21.
22.                   except KeyboardInterrupt :
```

```
23.                         break
24.                         EXIT_FLAG = 1
25.     if EXIT_FLAG == 1:
26.           break
```

Figure 5.11 : Code de la boucle d'obtention des données du système d'OptiTrack ③

Après avoir obtenu les données de localisation de la station Matlab, nous les transformons en données utilisables, puis construisons une matrice T avec tous les paramètres obtenus du système OptiTrack. Cette matrice est en fait la deuxième transformation que nous effectuons (drone → global). La première transformation est une transformation statique que nous avons définie au début de notre code avec nos variables globales (voir la figure 5.10). Nous multiplions ces deux matrices pour obtenir la transformation finale du système de la caméra jusqu'au système global (MAT_TRANSFO). Ceci est illustré à la figure 5.12.

```
 1.     if OPTITRACK_OVERRIDE == 0:
 2.           POSITIONX = float(parsedData[2])/XSCALING
 3.           POSITIONZ = float(parsedData[3])/ZSCALING
 4.           POSITIONY = -1 * float(parsedData[4])/YSCALING
 5.           ROLL = float(parsedData[5])*3.141596/180
 6.           YAW = float(parsedData[6])*3.141596/180
 7.           PITCH = float(parsedData[7])*3.141596/180
 8.           ca = math.cos(YAW)
 9.           sa = math.sin(YAW)
10.           cb = math.cos(-1*ROLL)
11.           sb = math.sin(-1*ROLL)
12.           cg = math.cos(PITCH)
13.           sg = math.sin(PITCH)
14.           matrice_transfo_optitrack = np.array([[ca*cb, ca*sb*sg -
              sa*cg, ca*sb*cg + sa*sg, POSITIONX],[sa*cb, sa*sb*sg +
              ca*cg, sa*sb*cg - ca*sg, POSITIONY],[-sb, cb*sg, cb*cg, PO-
              SITIONZ], [0, 0, 0, 1]])
15.           MAT_TRANSFO = np.dot(matrice_transfo_optitrack,
              matrice_transfo_static)
16.           #print MAT_TRANSFO
```

Figure 5.12 : Code de construction des matrices de transformations ④

Après avoir obtenu les matrices de transformation, nous devons passer au traitement d'image. Nous commençons par prendre une image et nous la convertissons en format HSV. Après la conversion, nous appliquons pour un filtre pour détecter la couleur rouge. Nous avons choisi le format HSV parce qu'il facilite le filtrage de l'image en fonction des

couleurs. Le format HSV contient trois champs : la teinte (hue), la saturation et la luminosité (value), telle qu'illustrée à la figure 5.13.

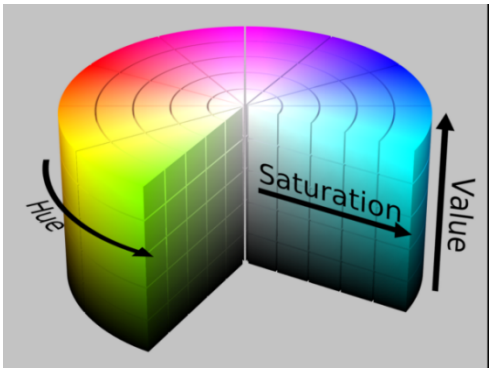


Figure 5.13 : Illustration du format HSV [6]

Figure 5.14 : Échelle du champ de la teinte [5]

Comme le démontre la figure 5.14, il suffit de prendre la teinte correspondant à la couleur rouge pour pouvoir filtrer notre image. Nous appliquons ensuite un filtre pour obtenir les pixels ayant un champ de teinte entre 350 et 10 degrés. Nous utilisons aussi un autre filtre pour éliminer les bruits. Le code de ces filtres est présenté à la section 5.15.

```
1.      _,frame = cap.read()
2.
3.      #red thresholding
4.
5.      hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
6.      lowerHUE = cv2.inRange(hsv,np.array((0, 50, 100)), np.array((10,
        255, 255)))
7.      upperHUE = cv2.inRange(hsv,np.array((160, 50, 100)),
        np.array((179, 255, 255)))
8.      redThresh = cv2.addWeighted(lowerHUE, 1.0, upperHUE, 1.0, 1.0,
        0.0)
9.
10.     #filtering edges to remove as much noise as possible
11.     redThresh = cv2.blur(redThresh,(6,6))
12.     thresh2 = redThresh.copy()
```

Figure 5.15 : Code pour l'application des filtres ⑤

Après avoir filtré nos couleurs, nous utilisons le détecteur de blob pour déterminer les agglomérations de couleurs dans notre image. Cela nous permet de déterminer où sont situées les cibles dans notre image. Nous obtenons en retour les couples de pixels représentant les centres de masses de toutes les cibles dans notre image, telle qu'illustrée par le code de la figure 5.16.

```
1.      #setting params for keypoint detection
2.      params = cv2.SimpleBlobDetector_Params()
3.      params.minThreshold = 240
4.      params.maxThreshold = 255
5.      params.filterByArea = False
6.      params.filterByCircularity = False
7.      params.filterByConvexity = False
8.      params.filterByColor = True
9.      params.blobColor = 255
10.
11.     #simpleBlob_detector to detect keypoints
12.     detector = cv2.SimpleBlobDetector_create(params)
13.     keypoints = detector.detect(thresh2)
14.     inputPoints = np.zeros((len(keypoints), 1, 2), dtype=np.float32)
15.
16.     #putting circles on center of blobs
17.     for i in range(len(keypoints)):
18.            cv2.circle(frame,(int(keypoints[i].pt[0]),int(keypoints
               [i].pt[1])), 5,255,-1)
19.            inputPoints[i,0,0] = keypoints[i].pt[0]
20.            inputPoints[i,0,1] = keypoints[i].pt[1]
21.
22.     #if there is a target....
```

Figure 5.16 : Code de détection des centres de masses avec le Blob Detector ⑥

Après avoir obtenus les centres de masses de nos cibles, nous allons corriger la distorsion les pixels avec les matrices des coefficients intrinsèques et de distorsion dans le but de générer un vecteur normalisé qui sera multiplié par notre matrice de transformation pour le mettre dans notre système de référence global. Ce code informatique est présenté à la figure 5.17.

```
1.      if len(keypoints) > 0:
2.
3.            #use the center of blobs and undistort it to get normalized
              pixels
4.            #(0 -> 1     for z = 1)
```

```
5.            outputPoints = cv2.undistortPoints(inputPoints, K_Mat, kc)
```

Figure 5.17 : Code de la correction de la distorsion des cibles ⑦

La prochaine étape, dont le code est présenté à la figure 5.18, consiste à projeter le vecteur sur la carte d'élévation. Nous commençons par prendre les vecteurs des cibles et nous les multiplierons par les matrices de transformations pour les projeter sur la carte d'élévation.

```
1.    #calculate the origin (center of the camera in the OptiTrack Ref-
      erence frame)
2.    origine = np.dot(MAT_TRANSFO, np.array([[0], [0], [0], [1]]))

3.    #os.system('clear')
4.    #for each target..
5.    for i in range(len(keypoints)):
6.          outputPoints[i,0,1] = (-1)*outputPoints[i,0,1]
7.          #create a vector and a walking point
8.          cibles = np.array([[outputPoints[i,0,0]],
            [outputPoints[i,0,1]], [-  1], [1]])
9.          vecteurcible = np.dot(MAT_TRANSFO, cibles)
10.         position = origine
11.         vecteur_direction = vecteurcible-origine
12.         #walk along vector until you hit the digital
13.         #élévation map
```

Figure 5.18 : Code pour la multiplication des vecteurs par la matrice T ⑧

Pour y parvenir, nous utilisons la position de notre drone comme origine. Ensuite, nous appliquons successivement le vecteur de la cible jusqu'à tant que celle-ci fasse intersection avec la carte d'élévation. Lorsque toutes ces opérations sont complétées, nous affichons l'image avec toutes les cibles identifiées, marquées d'un cercle bleu en leur centre, et nous envoyons la position X et Y, ainsi que son élévation à la station de contrôle. Le code informatique de cette étape finale est reproduit à la figure 5.19.

```
1.    while(position[2,0] > Elevation[int(position[0,0] +
      XOFFSET)][int(position[1,0] + YOFFSET)]):
2.          position = position + vecteur_direction
3.          #print the x and y coordinates of the exact location on the
            Digital map/OptiTrack reference
4.
5.          POS = 'Cible Trouver !! Position X :' + repr(position[0,0])
            + '
6.          Position Y :' + repr(position[1,0]) + 'Élévation : ' +
            repr(position[2,0])
```

```
 7.            print(POS)
 8.
 9.      #Show it, if key pressed is 'Esc', exit the loop
10.      cv2.imshow('frame',frame)
11.      cv2.imshow('thresh',thresh2)
12.      if cv2.waitKey(33)== 27:
13.            break
14.            # Clean up everything before leaving
15.            cv2.destroyAllWindows()
16.            cap.release()
17.            ser.close()
```

Figure 5.19 : Code pour la projection sur la carte d'élévation et l'envoi des données ⑨

5.3.4 Station de contrôle de l'opérateur

La station de contrôle de l'opérateur est simplement un ordinateur ayant la capacité d'établir une communication sécurisé vers notre drone afin que l'opérateur puisse lancer une mission, en vérifier les résultats, extraire au besoin la photographie d'une cible ou tout simplement effectuer des opérations de maintenance sur le drone. Bien que ne faisant pas implicitement partie de la station de contrôle de l'opérateur, nous utilisons un routeur sans fil afin d'agir en tant que point d'accès et de gestionnaire des adresses IP.

5.3.4.1 Station de contrôle de l'opérateur – spécifications et contraintes

Afin de faciliter la communication avec le Raspberry Pi qui fonctionne sous Raspian, une version modifiée de Linux Debian, nous avons installé la version Jessie de Linux Debian sur notre station. Notre station est également équipée d'un adaptateur réseau sans fil (Wifi). Il n'existe pas vraiment de contrainte quant à la configuration de la station de l'opérateur, nous avons souvent utilisé nos ordinateurs personnels avec différentes configurations et même différents systèmes d'exploitation et nous n'avons éprouvé aucun problème à se connecter au Raspberry Pi. Afin de pouvoir utiliser toutes les fonctions que nous avons développées sur le Pi, il est préférable que l'ordinateur établissant la connexion puisse supporter le système de partage d'affichage graphique VNC, ce qui ne

pose aucun problème sur les systèmes d'exploitation Linux ou BSD.

5.3.4.2 Station de contrôle de l'opérateur – conception et fonctionnement

Aucune conception n'a été réalisée pour la station de contrôle de l'utilisateur. Nous utilisons le système VNC afin de transmettre l'affichage graphique du Raspberry Pi sur un ordinateur distant. Ce système de partage d'affichage n'est cependant pas sécurisé, ce qui va à l'encontre d'une des exigences que nous nous étions fixées au début du projet. Il existe une version sécurisée de VNC qui pourrait être implanté pour un développement futur.

Au début de la conception de notre projet, nous avions configuré le Pi pour agir en tant que point d'accès pour créer un réseau ad hoc et gérer directement ses connexions. Cette façon de faire fonctionnait très bien et apportait une grande stabilité, mais nous devions enlever cette configuration à chaque fois que nous avions besoin de faire une mise à jour du Pi nécessitant une connexion à internet. L'utilisation d'un routeur sans fil auquel la station de contrôle et le Pi se connecte automatique permet d'éviter ses tracas, sans sacrifices important à la stabilité de la connexion.

5.4 Description des interfaces

Notre conception comprend trois interfaces afin de permettre aux quatre modules de communiquer entre eux. Il s'agit respectivement des interfaces OptiTrack – Matlab, Matlab – Raspberry Pi et Raspberry Pi – Station de l'opérateur. Elles sont décrites dans les sections qui suivent.

5.4.1 Interface OptiTrack – MATLAB

L'application NatNet, fournie par OptiTrack, est installée sur notre station MATLAB. Il contient par défaut des fonctionnalités pour communiquer à l'aide de sockets TCP. Comme nous l'avons décrit dans la section du fonctionnement de la station Matlab, notre script établit la communication avec le serveur OptiTrack et formule des demandes afin d'obtenir les informations de localisation les plus récentes. Nous avons utilisé l'exemple de script fourni par OptiTrack, aucune modification, à l'exception d'indiquer l'adresse IP du serveur OptiTrack et le nom des variables dans laquelle enregistré les valeurs obtenues n'a été nécessaire. Ces données sont la localisation du drone dans l'arène OptiTrack à l'aide de ces coordonnées x, y et z ainsi que les quatre quaternions qx, qy, qz et qw qui permettent de déterminer son inclinaison dans les trois axes.

5.4.2 Interface MATLAB – Raspberry Pi

Nous avons développé cette interface afin d'assurer le transfert des informations de localisation obtenu par la station Matlab vers le Raspberry Pi. Comme nous l'avons décrit dans la section concernant la station Matlab, l'information de localisation est transmise dans une chaine de caractère diffusé à intervalles réguliers, environ une fois par seconde, via les antennes XBee. Nous avons choisi d'utiliser les antennes XBee, plutôt que le réseau Wifi déjà existant, pour transmettre ces données afin de les isoler et de réellement simuler le travail qu'effectuerait un système GPS si nous étions à l'extérieur. Les antennes doivent être associées à leur première utilisation à l'aide des instructions du manufacturier, par la suite, aucune autre opération n'est nécessaire afin qu'elles communiquent entre elles. Elles utilisent un protocole de communication sécurisé, nous n'avons qu'à leur soumettre notre chaîne de caractère, celle-ci est encryptée, transmises,

puis vérifiée et décryptée par l'antenne réceptrice. L'information devient à ce moment disponible pour le Raspberry Pi. Le format de la chaîne de caractère transmise est présenté à la figure 5.20.

```
1.      DEBUT ID[identification_numérique_de_la_séquence] [coordX]
        [coordY] [CoordZ] [angle_lacet] [angle_Tangage] [angle_Roulis] FIN
```

Figure 5.20 : Format de la chaîne de caractère des données de localisation

Les informations en italique entre crochets sont générées par le script Matlab. Le numéro de séquence est un nombre numérique incrémenté de 1 à chaque envoi pour différencier les différents envois. Les balises DEBUT et FIN sont utilisées par le Raspberry Pi afin de vérifier la validité de la chaîne de caractère reçue.

5.4.3 Interface Raspberry Pi – Station de l'opérateur

La station de l'opérateur permet à celui-ci d'entrer en communication avec le Raspberry Pi via une connexion wifi sécurisée. Comme décrit dans la section sur le fonctionnement de la station l'opérateur, un routeur sans fil est utilisé afin de gérer les adresses IP et les connexions. La connexion est effectuée à l'aide du protocole SSH ou VNC existant. Il permet d'établir une connexion simplement et sécurisé.

6. Identification de l'équipement

Cette section présente les équipements de notre système. Le logiciel du Raspberry Pi pour l'identification de cible et de calcul de position a été entièrement développé par nous. Le script Matlab est une adaptation d'un exemple fourni par OptiTrack, auquel nous avons ajouté une capacité de diffusion à l'aide des antennes XBee. La maquette de support est un modèle de drone à imprimer en 3D que nous avons trouvé sur internet et adapté, cependant n'importe quel support pour le boitier du Pi pourrait être utilisé. Le

convertisseur doit fournir une alimentation continue et stable de 5 volts au Pi, selon le voltage fourni par le bloc pile.

Éléments développés :	**Modèle / no de pièce**	**Détails**
Logiciel de recherche de cible et de calcul de position du Raspberry Pi		Langage Python
Script de conversion des données OptiTrack et de diffusion sur antenne XBee		Langage Matlab
Maquette de support des équipements		
Convertisseur CC 11V ⎓ 5V		Input : selon pile Output : 5V
Éléments standards :		
Raspberry Pi	Modèle 2B	
Carte MicroSD	16 Gig	
Raspberry Pi Camera	Modèle 1.3	
Antenne wifi USB	Générique	Nombre : 2
Antenne XBEE USB	S2	Nombre : 2
Bloc pile 11 V + chargeur	Générique	
Routeur	Générique	
Équipement de soutien :		
Système OptiTrack		Laboratoire
Station Matlab	Windows 7 Matlab R2014A	Laboratoire
Station Linux avec antenne wifi	Debian	

Tableau 6-1 : Liste des équipements

7. Résultats

Dans cette section, nous débutons par expliquer la méthodologie de vérification utilisée pour nos essais principaux. Puis, à la section 7.2, nous présentons et analysons les résultats de ces essais. Par la suite, à la section 7.3, nous révisons chacune des exigences que nous avions établies au début du projet en fonction des résultats obtenus lors des

essais principaux. Toujours à la section 7.3, lorsque pertinent, les résultats d'essais secondaires sont présentés et des commentaires accompagnent chacune des exigences. À la section 7.4, nous extrapolons nos résultats afin de ramener notre projet à l'échelle 150 :1 tel quel décrite dans la portée du projet à la section 1.3. Finalement, la section 7.5, nous approfondissons l'analyse des résultats obtenus et nous présentons des pistes de solutions qui pourraient être développées afin d'améliorer le projet.

7.1 Test

Afin d'évaluer les performances de notre système de repérage aéroporté, nous avons développé une série de tests ayant pour objectif de vérifier si notre projet respecte la mission que nous avions énoncé à l'étape de planification du projet. Ces tests permettent également de vérifier l'atteinte ou non de la majorité des exigences que nous nous étions fixés lors de l'énoncé des besoins [4].

Les tests 1, 2 et 3 consistent à identifier 10 cibles (7 dans le cas du test 3) disposées dans le champ de vision de la maquette de notre drone. Les cibles sont demeurées au même endroit pour les trois séries de lectures, mais le drone a été déplacé et son inclinaison changée. Dans le cas du test 3, le drone fut placé directement au-dessus des cibles, aussi parallèle que possible avec le sol, ce qui explique que nous ne pouvions plus capter toutes les cibles dans son champ de vision. Nous avons disposé les cibles de sorte qu'elles soient à différentes élévations, que quatre d'entre elles soient espacées d'un centimètre et que deux d'entre elles aient une surface inférieure à 8 cm^2. Le drone demeure immobile durant la collecte de données.

Chacun des trois tests a été réalisé de la façon selon les modalités qui suivent. Nous avons premièrement noté les mesures exactes des cibles que nous avions disposées dans

l'arène. Nous avons ensuite pris une lecture à l'aide de notre système de repérage. Puis, nous avons vérifié si nous étions en mesure d'identifier toutes les cibles. Finalement, nous avons mesuré l'écart entre leurs localisations réelles et celles calculées par notre système. Les résultats sont présentés et analysés à la section 7.2.

Les tests 4, 5 et 6 consistent à évaluer la justesse des lectures lorsque notre drone se déplace en ligne droite à différentes vitesses. Nous avons utilisé une cible d'une grandeur de 200 cm^2 à une altitude de 0 cm (plancher) et nous l'avons survolée avec notre drone à une altitude de 100 cm se déplaçant sur l'axe des X, positif vers négatif, à des vitesses de 7, 13 et 26 cm/sec respectivement. Nous avons effectué 3 séries de lectures pour chaque vitesse et nous avons calculé la moyenne et l'écart type par rapport à la localisation réelle de la cible.

D'autres vérifications furent aussi nécessaires afin de valider certaines exigences de notre projet, celles-ci sont expliquées à la section 7.3 lors de l'analyse individuelle des exigences.

7.2 Résultats

Les tableaux 7-1, 7-2 et 7-3 présentent les données recueillies lors des trois premiers tests. Les données y sont présentées pour chacune des 10 cibles (7 dans le cas du test 3) en fonction de chacun des plans de localisation en x, en y et en z. L'unité de mesure des distances de ces données est le centimètre. La localisation du drone y est indiquée en centimètre et son inclinaison en degrés.

Position du drone (angles en degrés):

x (cm):	49	y (cm):	39	z (cm):	172
roulis (deg)	-32.8	tangage (deg)	-38.2	lacet (deg)	-39.8

	Réel			Mesuré			Erreur absolu		
# cible	x (cm)	y (cm)	z (cm)	x (cm)	y (cm)	z (cm)	x (cm)	y (cm)	z (cm)
1	-15	162	75	-10	165	75	5	3	0
2	-26	163	75	-24	167	75	2	4	0
3	-47	108	0	-50	112	0	3	4	0
4	-70	136	0	-75	139	0	5	3	0
5	-74	93	0	-81	95	0	7	2	0
6	-104	138	32	-109	144	32	5	6	0
7	-100	143	32	-106	150	32	6	7	0
8	-102	142	32	-112	150	32	10	8	0
9	-105	155	32	-107	160	32	2	5	0
10	-148	187	69	-152	199	69	4	12	0

Tableau 7-1 : Données du test 1

Dans ce premier test, notre drone était placé relativement éloigné des cibles avec une forte inclinaison. Nous avons obtenu des écarts relatifs variant de 2 cm à 12 cm. L'écart en z est toujours de zéro, puisque l'altitude mesurée correspond à celle de la carte d'élévation préalablement chargée en mémoire.

Position du drone (angles en degrés):

x (cm):	-5	y (cm):	-22	z (cm):	161
roulis (deg)	-32.6	tangage (deg)	-46.0	lacet (deg)	92.0

	Réel			Mesuré			Erreur absolu		
# cible	x (cm)	y (cm)	z (cm)	x (cm)	y (cm)	z (cm)	x (cm)	y (cm)	z (cm)
1	-15	162	75	-10	156	75	5	6	0
2	-26	163	75	-23	158	75	3	5	0
3	-47	108	0	-48	109	0	1	1	0
4	-70	136	0	-73	135	0	3	1	0
5	-74	93	0	-78	94	0	4	1	0
6	-104	138	32	-104	140	32	0	2	0
7	-100	143	32	-101	146	32	1	3	0
8	-102	142	32	-107	147	32	5	5	0
9	-105	155	32	-103	156	32	2	1	0
10	-148	187	69	-145	193	69	3	6	0

Tableau 7-2 : Données du test 2

Dans le cadre du test 2, nous avons rapproché notre drone des cibles, tout en conservant une inclinaison assez forte. Les résultats obtenus lors de ce test sont plus précis que ceux obtenus durant lors du premier test, il est probable que cette précision plus grande soit en partie expliquée par le fait que notre drone se situait plus près des cibles. Les écarts absolus varient de 0 à 6 cm.

Position du drone (angles en degrés):

x (cm):	-58	y (cm):	117	z (cm):	156
roulis (deg)	-5.3	tangage (deg)	-0.8	lacet (deg)	-87.0

	Réel			Mesuré			Erreur absolu		
# cible	x (cm)	y (cm)	z (cm)	x (cm)	y (cm)	z (cm)	x (cm)	y (cm)	z (cm)
1	-15	162	75	***	***	***			
2	-26	163	75	***	***	***			
3	-47	108	0	-45	109	0	2	1	0
4	-70	136	0	-67	137	0	3	1	0
5	-74	93	0	-76	95	0	2	2	0
6	-104	138	32	-96	149	32	8	11	0
7	-100	143	32	-101	150	32	1	7	0
8	-102	142	32	-99	144	32	3	2	0
9	-105	155	32	-98	160	32	7	5	0
10	-148	187	69	***	***	***			

*** cibles non-visibles

Tableau 7-3 : Données du test 3

Lors du test 3, nous avions placé le drone directement au-dessus des cibles aussi parallèle que possible avec le plancher. Nous avons dû éliminer certaines cibles de l'analyse puisqu'il était impossible de les inclure dans le champ de vision du drone. Les cibles éliminées portent la mention ***. Encore une fois, les données obtenues sont précises, bien que légèrement moins que celle du test 2. Le plus grand écart est de 11 cm.

Les tableaux 7-4, 7-5 et 7-6 présentent les données recueillies lors des tests 4, 5 et 6 lorsque le drone était en déplacement à des vitesses respectives de 7 cm/s, 13 cm/s et 26

cm/s. Trois essais ont été faits pour chaque vitesse. La variation du nombre de lectures par essai est attribuable au temps d'exposition des cibles, à plus grande vitesse, moins de lectures ont le temps d'être captées avant que les cibles ne disparaissent du champ de vision du drone.

Déviation (en cm) par rapport à l'emplacement réel de la cible, v=7cm/s

Essai 1	
X	Y
19	6
14	4
20	1
24	6
20	3
18	42
14	5

X moy	Y moy
19	10

Essai 2	
X	Y
23	7
3	7
7	4
14	6
19	6
15	7
17	2
14	5

X moy	Y moy
14	5

Essai 3	
X	Y
24	6
12	7
15	5
17	8
21	4
22	7
14	7

X moy	Y moy
18	6

Tableau 7-4 : Données du test 4

Les lectures des trois essais du test 4(tableau 7-4) ont été prises lorsque le drone se déplaçait à une vitesse de 7 cm/s. Nous pouvons remarquer qu'il n'y a pas une très grande variation des moyennes d'écarts entre chaque série de lecture, cependant l'imprécision quant à la localisation réelle de la cible et la localisation détectée par le drone est plus grande que lors des essais réalisés lorsque notre drone était immobile.

Déviation (en cm) par rapport à l'emplacement réel de la cible, v=13cm/s

Essai 1	
X	Y
24	13
20	4
14	6
21	9
22	3
18	7

X moy	Y moy
20	7

Essai 2	
X	Y
18	7
23	5
19	6
18	6
27	5

X moy	Y moy
21	6

Essai 3	
X	Y
24	4
24	3
20	8
17	8
25	5

X moy	Y moy
22	5

Tableau 7-5 : Données du test 5

Les lectures des trois essais du test 5(tableau 7-5) ont été prises lorsque le drone se déplaçait à une vitesse de 13 cm/s, soit le double de la vitesse du test 4. Nous pouvons remarquer, encore une fois, qu'il n'y a pas une très grande variation des moyennes d'écarts entre chaque série de lectures. Cependant, les écarts sont plus grands que lors du test 4, possiblement dû à l'augmentation de la vitesse de déplacement du drone.

Déviation (en cm) par rapport à l'emplacement réel de la cible, v=26cm/s

Essai 1	
X	Y
39	6
40	4
33	6

X moy	Y moy
37	5

Essai 2	
X	Y
31	7
24	7
38	9
27	9

X moy	Y moy
30	8

Essai 3	
X	Y
39	10
29	8
28	7
29	6

X moy	Y moy
31	8

Tableau 7-6 : Données du test 6

Les lectures des trois essais du test 6(tableau 7-6) ont été prises lorsque le drone se déplaçait à une vitesse de 26 cm/s, soit le quadruple de la vitesse du test 4. Les écarts

entre les moyennes consignées et les localisations réelles des cibles sont maintenant beaucoup plus grands que lorsque le drone est immobile ou se déplace lentement.

Notre série de tests a permis de démontrer que notre système pouvait identifier des cibles et déterminer leurs localisations avec précision lorsque notre drone était dans un état stationnaire. Nous avons aussi pu démontrer que nous sommes en mesure d'identifier une cible lorsque notre drone se déplace, cependant, plus sa vitesse augmente, plus nous perdons de la précision quant au calcul de la localisation réelle d'une cible.

7.3 Vérification des exigences

Le tableau 7-7 énumère les principales exigences fonctionnelles que nous avions incluses dans notre énoncé des besoins [4] ainsi qu'une mention pour chacune de ces exigences concernant l'atteinte ou non de celle-ci. À la suite du tableau, chaque exigence est reprise, accompagnée de son descriptif et de commentaires relatifs à sa vérification.

Nom de l'exigence	Atteinte / Non-atteinte
FR-01 : Prise de Photo	Atteinte
FR-02 : Identification d'une cible	Atteinte
FR-03 : Calcul de la localisation d'une cible	Atteinte
FR-04 : Communication des résultats	Atteinte
FR-05 : Alimentation sécurisée	Atteinte
FR-06 : Sauvegarde des informations	Atteinte
FR-07 : Relief du terrain	Atteinte
FR-08 : Indépendance du système	Atteinte
IR-01 : Communication avec le système OptiTrack	Atteinte
PR-01 : Fréquence d'analyse d'image	Atteinte
PR-02 : Durée de mission	Atteinte
PR-03 : Champs de vision	Atteinte
PR-04 : Taille minimale de la cible	Atteinte
PR-05 : Résolution de repérage	Atteinte
PR-07 : Sécurisation du système	Non-atteinte

Tableau 7-7 : Liste de contrôle des atteintes de performances

Comme nous pouvons le remarquer au tableau 7-7, 14 des 15 exigences que nous nous

étions fixées ont été atteintes. Voici quelques commentaires

FR-01 : Prise de Photo – Le système de repérage doit être en mesure de photographier les cibles.

Notre système est en mesure de capter des images. Lors de l'exécution de notre programme, nous affichons à l'écran en temps réel ce que le Raspberry Pi voit. Un exemple de capture d'image est présenté à la figure 7-8 (gauche). Cette exigence est rencontrée.

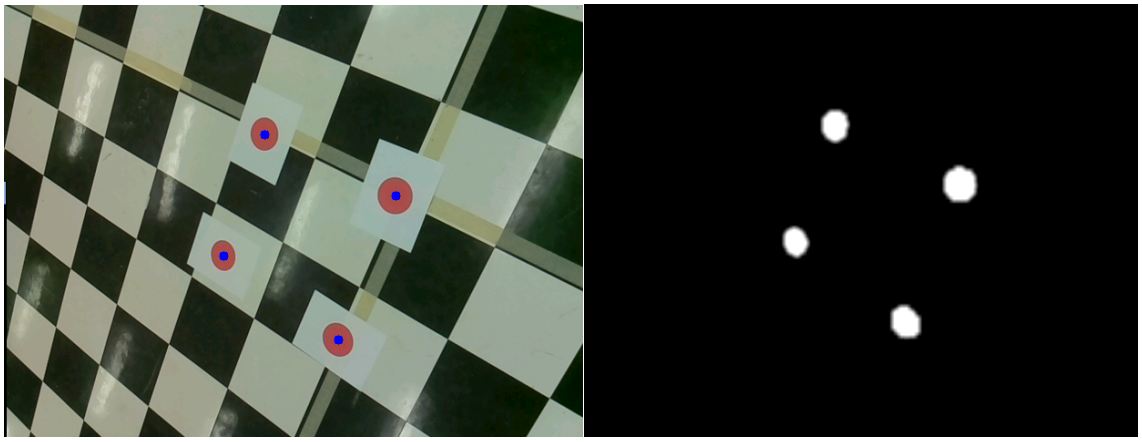


Figure 7-8 : Exemple de capture d'image

FR-02 : Identification d'une cible – Le système de repérage doit être en mesure d'analyser les informations recueillies afin d'identifier une cible déterminée par l'opérateur.

Lors de son exécution, notre programme identifie non seulement les cibles (tel que démontré à la figure 7-8 (droite)), mais il indique le centre de celle-ci à l'aide d'un point de couleur bleue (figure 7-8 (gauche)). Cette exigence est rencontrée.

FR-03 : Calcul de la localisation d'une cible – Le système doit être en mesure de calculer la localisation sur les axes x, y, z de l'arène d'une cible au sol selon les exigences de

résolution précisées dans les exigences de performance.

Lors de nos essais de vérification, notre programme nous a retourné la localisation des cibles que nous avions déposées au sol. Le détail de ces essais est décrit en détail à la section 7-2, la position en x, y et z de chaque cible a été retournée pour chacun des essais. Cette exigence est rencontrée.

FR-04 : Communication des résultats – Le système doit être en mesure de transmettre à l'opérateur la localisation des cibles identifiées.

Lors de l'exécution de notre programme, la localisation des cibles identifiées est affichée et mise à jour continuellement. La figure 7-9 montre un exemple de données retournées par le Raspberry Pi. Cette exigence est rencontrée.

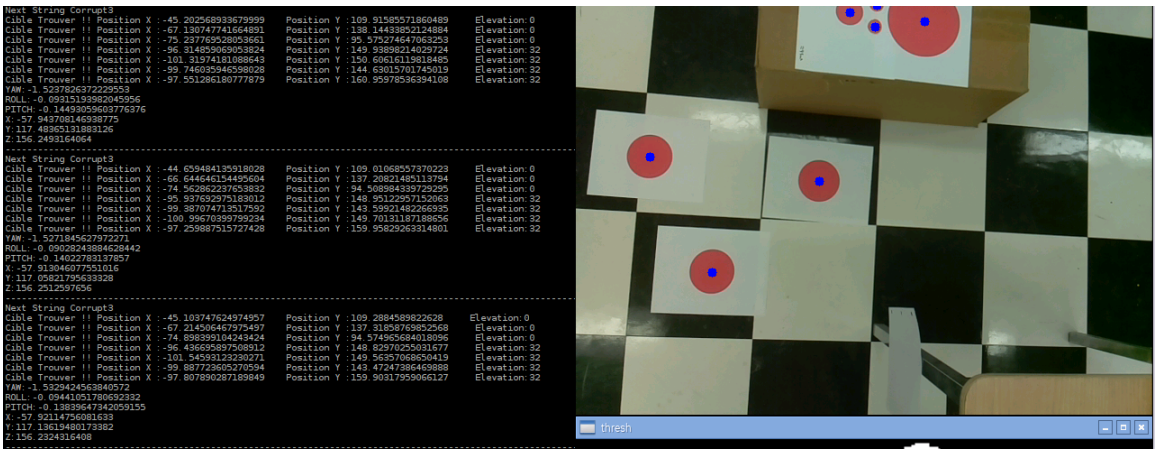


Figure 7-9 : Exemple d'affichage des données de localisation

FR-05 : Alimentation sécurisée – Le système d'alimentation doit fournir une tension de 5 volts, avec un écart maximal de 10% (spécifications du Raspberry Pi), de plus, il doit être doté d'une protection contre les pointes de tension et prévenir l'utilisateur en cas de basse tension.

Lors de la conception du convertisseur CC, nous avons utilisé un régulateur de tension LM323K. Le manufacturier du composant garantit une précision de ± 1%. Selon les vérifications effectuées avec un multimètre, nous avons mesuré des tensions situées à l'intérieur ce cet écart de 1%, situé entre 5.03 et 4.98 volts. Cette exigence est rencontrée.

FR-06 : Sauvegarde des informations – Le système doit être en mesure de garder les photos et les coordonnées des cibles repérées en mémoire.

Notre système comprend une fonction de sauvegarde des images captées et des détails de leurs localisations chaque fois qu'au moins une cible est détectée. Cependant, nous désactivons cette fonction par défaut. Parce que notre vitesse typique d'analyse se situe à environ une image par seconde, la mémoire limitée de notre Raspberry Pi se remplit très rapidement. La figure 7-10 montre un exemple de sauvegarde d'un fichier photo et d'un fichier texte pour quelques séquences de détections. Cette exigence est rencontrée.

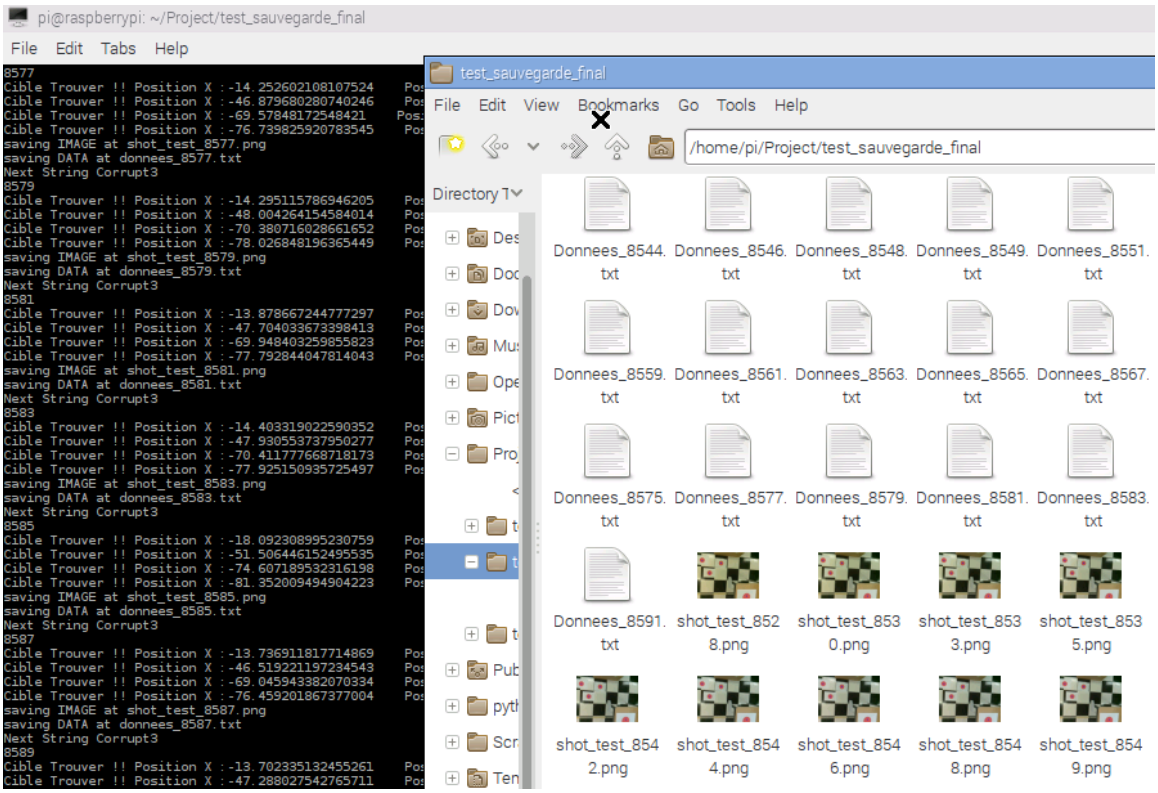


Figure 7-10 : Sauvegarde des cibles détectées

FR-07 : Relief du terrain– Le système doit tenir en compte du relief du terrain dans le calcul de la position de la cible à l'aide d'une carte topographique préenregistrée.

Notre concept prend un compte une carte d'élévation préalablement sauvegardée dans la mémoire du Pi et chaque position calculée dans nos essais tenait compte de l'élévation des cibles. Ces résultats ont été analysés plus en profondeur dans la section 7.2. Cette exigence est rencontrée.

FR-08 : Indépendance du système – Le système doit être capable d'opérer de manière autonome une fois la mission lancée.

Une fois la mission lancée, notre programme s'exécute de façon autonome, sans l'intervention d'un opérateur. Cette exigence est rencontrée.

IR-01 : Communication avec le système OptiTrack – notre système devra être en mesure de recevoir les informations de localisation transmises par le réseau OptiTrack.

Notre système est en mesure de recevoir les coordonnées du système OptiTrack via l'antenne XBee. En fait, notre programme principal ne fonctionnerait tout simplement pas si nous ne recevions pas ces informations, car le calcul de la localisation d'une cible serait impossible sans connaître la localisation et l'inclinaison du drone. Cette exigence est rencontrée.

PR-01 : Fréquence d'analyse d'image – Le système doit être en mesure d'analyser une nouvelle image à toutes les secondes.

Notre vitesse d'analyse d'image varie en fonction de la vitesse à laquelle une coordonnée de localisation est reçue de la station Matlab, de façon générale, nous travaillons avec une vitesse d'environ une analyse par seconde. À la figure 7-11, nous pouvons voir un chiffre représentant le délai en seconde sous chaque série de cibles détectées. Cette exigence est rencontrée.

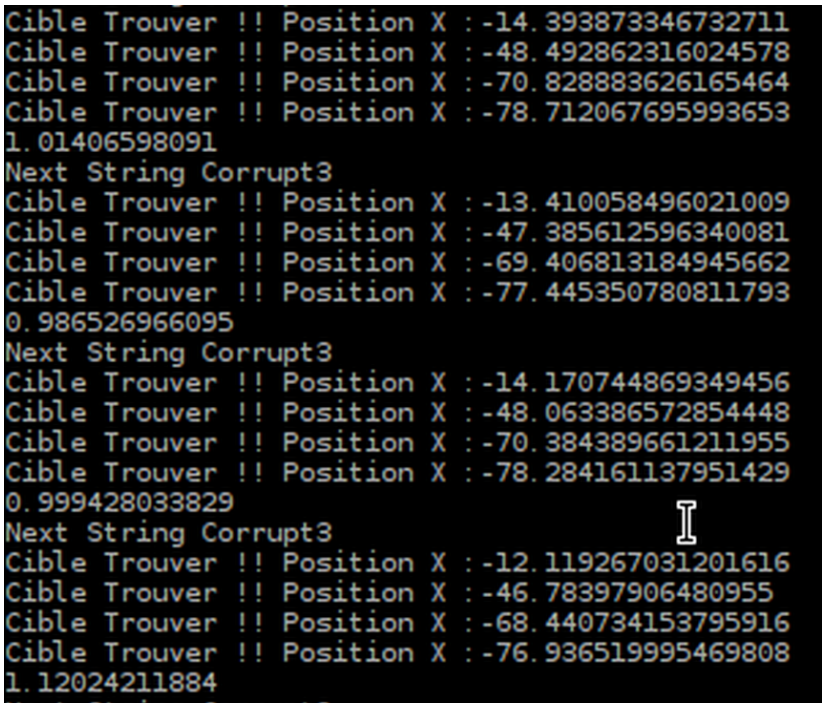


Figure 7-11 : Calcul de la fréquence d'analyse

PR-02 : Durée de mission – Le système devra être en mesure d'opérer pendant un minimum de 4 heures avec une source d'énergie autonome.

Le Raspberry Pi est l'élément qui dépend de cette source d'énergie et qui doit fonctionner de façon autonome. Nous avons utilisé notre Raspberry Pi à plusieurs occasions pour de longues périodes sans souffrir d'interruption d'alimentation, en fait, durant nos tests, la période la plus courte d'opération sur une même pile a été de 8 heures. Cette exigence est rencontrée.

PR-03 : Champs de vision - Le système de repérage devra être capable de détecter une cible terrestre lorsque celle-ci se retrouve entièrement dans son champ de vision.

Tel que nous l'avons démontré lors de nos essais à la section 7-2 et tel que nous en avons discuté à l'analyse de l'exigence FR-02, le simple fait que notre système puisse

automatiquement détecter une cible à l'intérieur de son champ de vision fait en sorte que nous avons atteint cet objectif. Cette exigence est rencontrée.

PR-04 : Taille minimale de la cible - Le système de repérage devra être en mesure de détecter n'importe qu'elle cible circulaire qui a plus de 45cm² de surface continue.

Lors des essais de la section 7-2, nous avons évalué différentes grosseurs de cibles circulaires. Le système n'avait aucune difficulté à repérer des cibles de huit centimètres carrés de surface. Lorsque la surface des cibles était inférieure à huit centimètres carrés, le système ne les détectait pas systématiquement. Les cibles de surface inférieures à 1 cm^2 ne sont pas détectées. Cette exigence est rencontrée.

PR-05 : Résolution de repérage – Le système de repérage devra être en mesure de repérer au moins 5 cibles différentes avec une distance minimale de 10 cm entre les cibles et de conserver une image de chacune de ces cibles.

Théoriquement, notre système est en mesure de détecter un nombre illimité de cibles à l'intérieur de son champ de vision, en autant qu'elles ne se chevauchent pas. Lors des essais décrits à la section 7-2, nous travaillions avec 10 cibles qui étaient toutes systématiquement détectées. Nous avions aussi réduit l'espacement entre certaines de ces cibles à moins d'un centimètre de séparation et nous parvenions toujours à les détecter distinctement. Cette exigence est rencontrée.

PR-07 : Sécurisation du système : Les communications avec l'opérateur du système doivent être encryptées.

À l'origine, nous avions prévu utiliser le protocole de communication SSH, un mode de communication sécurisé, entre le Raspberry Pi et la station de l'utilisateur. Nous avons cependant éprouvé des difficultés avec l'affichage de l'image de la caméra. Nous avons

donc modifié notre système afin d'utiliser la plateforme VNC qui permet d'afficher l'affichage d'un ordinateur sur une station distance. Ce mode de communication n'est pas sécurisé, bien qu'il existe une version sécurisée de VNC, nous n'avons pas eu le temps de l'implanter. Cette exigence n'est pas rencontrée.

7.4 Extrapolation des résultats

Pour mettre les résultats en perspective, il ne faut pas oublier que notre projet a été réalisé à une échelle réduite. L'échelle que nous avons utilisée est de 150:1. Nous avons choisi cette échelle, car c'est le ratio entre le plafond d'opération pour un drone du type auquel notre projet est destiné et le plafond de notre laboratoire de robotique. Afin d'illustrer la pertinence de nos résultats, prenons pour exemple la recherche d'un radeau de survie ayant une surface de visible de 4 mètres carrés, ce qui équivaudrait à un radeau de forme circulaire ayant un rayon de 1.12 mètre. En le réduisant à l'échelle du laboratoire, notre radeau aurait maintenant un rayon de 0.75 cm, c'est-à-dire une surface de 1.77 cm^2 dans le cas où le drone volerait à son plafond maximum. Lors de nos tests, nous avons réussi à détecter des cibles allant jusqu'à 1 cm^2, bien que leur détection n'était pas systématique. À une altitude de vol légèrement plus basse, un drone équipé de notre système n'aurait aucune difficulté à détecter toutes les cibles. Les surfaces de 8 cm^2 étaient systématique détectées, ce qui équivaudrait à un altitude de vol de 70 à 150 mètres pour notre drone selon les paramètres préalablement établis. Nous pouvons donc affirmer que nous avons atteint la vision de notre projet.

Un autre paramètre intéressant à analyser est la précision de détection de cibles multiples. En regardant les données des tests 1, 2 et 3, on peut estimer que la valeur moyenne des écarts de localisation entre le centre réel de la cible et celui calculé par notre système se situe entre 4 et 6cm. Lorsque l'on remet les choses en perspectives, en utilisant notre échelle de 150 :1, nous obtenons des déviations de 6 à 9 mètres. Ces données semblent vraiment élevées, par contre, si, lors d'une opération de recherche et sauvetage, un hélicoptère est envoyé à une distance de neuf mètres de la victime, il sera assurément en mesure de lui porter rescousse. De plus, dans nos tests nous étions capables de détecter une multitude de cibles sans avoir un impact significatif sur la performance du système. Ce qui rend notre système un outil idéal dans un scénario de recherche et sauvetage avec plusieurs victimes.

Par contre, il y a quand même certaines choses qui ne sont pas parfaite. Comme vous pouvez le voir dans les résultats du test 4,5 et 6, notre déviation augmente de manière quand même assez alarmante. Les plus grandes variations se trouvent dans le test 6 ou nous avons eu dans une de nos essaies, une moyenne d'environ 30 cm comme variation selon l'axe des X. Comme vous avez aussi pu le constaté, la variation augmentait principalement selon l'axe des X, l'axe sur lequel on se déplaçait pour effectuer nos tests. Si on retourne à l'échelle réelle, une déviation de 30cm constitue environ à une déviation de 45m, qui est selon nous encore considéré une marge acceptable, mais qui rend les résultats beaucoup moins intéressants.

7.5 Analyse des résultats

Nous croyons que les résultats que nous avons obtenus sont très prometteurs, particulièrement lorsque le drone est immobile. Il y a place à amélioration concernant les résultats obtenus lors des tests dynamiques, cependant, nous avons déterminé que la variation entre la position réelle et la position déterminée des cibles avait une forte corrélation avec la vitesse auquel le drone se déplaçait. Prenant ces faits en considération, nous croyons que la cause de ces écarts est probablement due à un délai significatif entre l'arrivée des données du système OptiTrack jusqu'à notre système. La figure 7.12 illustre cette problématique.

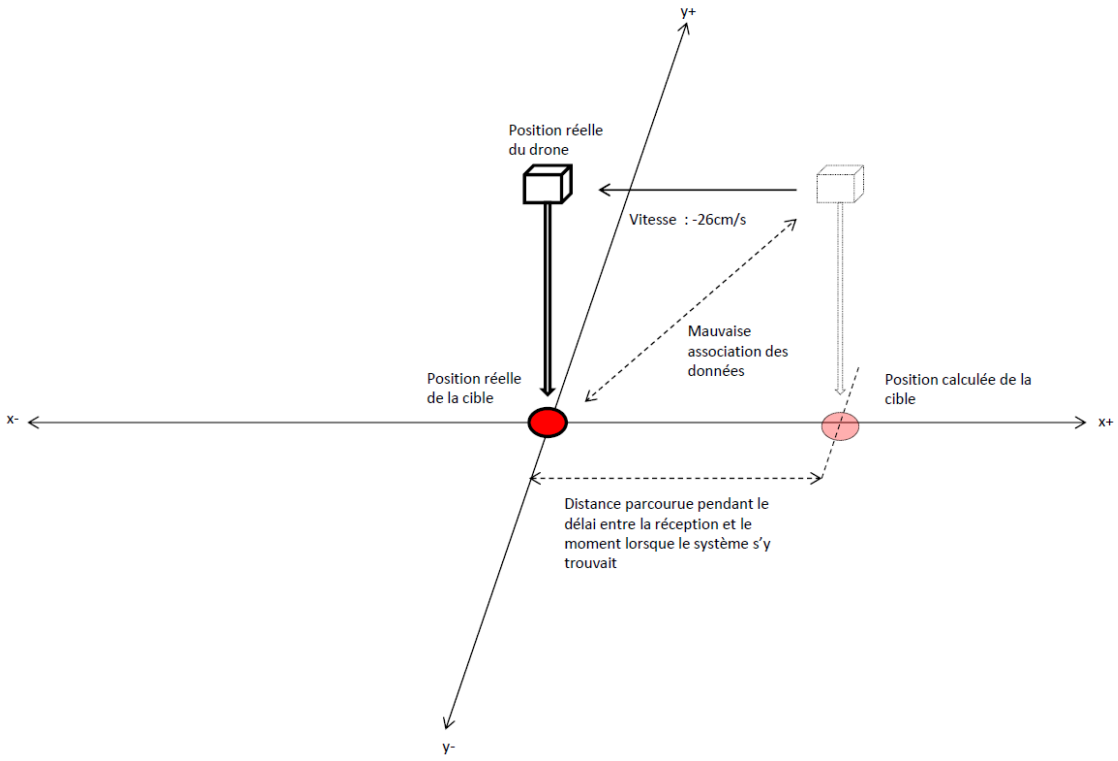


Figure 7.12 : Phénomène de délai

La source de ce problème résiderait dans le fait que nous utilisons plusieurs ordinateurs pour acquérir et traiter nos données. La majorité des délais proviennent des différentes interfaces, notamment celles entre le système OptiTrack et notre station Matlab, ainsi que celle entre la station Matlab et notre Raspberry Pi. Ces délais font en sorte que les informations obtenues par le Raspberry Pi sont obsolètes lorsqu'il les reçoit.

Par contre, une implémentation réelle du projet n'utiliserait pas le système OptiTrack, ni la station Matlab. Le système utiliserait probablement un vrai module GPS pour acquérir sa localisation et des gyroscopes afin de calculer ses inclinaisons. Cette façon de faire nous permettrait d'éliminer presque toutes les sources de délai entre l'obtention de ces données et la capture de l'image à analyser, rendant ici le calcul de la localisation de la cible beaucoup plus précise.

Une autre solution à ce problème serait l'utilisation d'accéléromètres et de gyroscopes afin de constamment mettre à jour la position et l'orientation du drone à partir d'une position initiale lors de son décollage. Le fait que ces modules soient rattachés directement au système réduirait le délai de transmission à une valeur très près de 0. Par contre, l'utilisation de ces modules ferait en sorte qu'on développerait une certaine imprécision après une longue utilisation et rendrait le système moins performant lors de très longues missions.

8. Sommaire

Dans le cadre de notre projet de créer un système de repérage aéroporté, nous avons

développé une application pouvant être exécutée sur un Raspberry Pi équipé d'une caméra. Nous avons ensuite installé cet équipement sur une maquette de drone auquel nous avions intégré un système d'alimentation comprenant une pile et un convertisseur CC. Nous avons finalement développé des outils afin de supporter la validité de notre projet, notamment une station Matlab nous permettant d'interroger un système de suivi d'objets OptiTrack et de retransmettre en temps réel la position de notre drone ainsi que son inclinaison.

Notre application débute une séquence d'analyse par la réception d'une capsule d'information de notre station Matlab comprenant la localisation du drone de notre arène de test selon un plan de référence x, y et z, ainsi que son inclinaison en roulis, tangage et lacet. Lorsqu'une séquence valide de données de localisation est reçue, une image est captée à l'aide de la caméra puis nous y recherchons une cible prédéterminée. Dans le cas de notre projet, nous avons identifié des cibles rouges en analysant la concentration de pixels de l'image obtenue. Lorsqu'une cible est détectée, son centre de masse est calculé, puis un vecteur entre le drone et la cible est généré. Ce vecteur est ensuite transformé en fonction de la position et de l'inclinaison du drone afin d'être projeté sur une carte d'élévation sauvegardée dans la mémoire du système. Cette opération nous permet de déterminer avec précision l'emplacement de la cible détectée. Cette information peut ensuite être relayé à l'opérateur ou sauvegardé en mémoire accompagnée d'une image de la cible.

Nous avons finalement vérifié l'efficacité de notre concept à l'aide d'essais. Une première série de tests nous a permis de déterminer que nous pouvions obtenir un haut degré de précision lorsque notre drone était dans une position statique. Nous avons

ensuite effectué une deuxième série de tests alors que notre drone était en mouvement. Nous parvenions toujours à systématique détecter nos cibles, par contre, nous perdions de la précision de localisation de nos cibles plus la vitesse de déplacement du drone augmentait. Nous avons établi que cette perte de précision pourrait grandement être améliorée si notre drone était responsable de déterminer lui-même sa position et son inclinaison, possiblement à l'aide d'un récepteur GPS et de gyroscopes, plutôt que de dépendre d'un système externe tel que nous avons utilisé dans le cadre de notre projet.

9. Conclusion

Ce document de conception détaillé avait pour objectif de présenter et de décrire un concept de système de repérage aéroporté que nous avons élaboré dans le cadre du cours de projet final de notre baccalauréat en génie électrique. Nous avions proposé de développer un équipement pouvant être embarqué sur un drone, opérant de façon autonome et ayant la capacité de détecter précisément une ou des cibles lorsque nous connaissions la position de notre drone et que nous disposions d'une carte d'élévation du secteur. Pour chaque cible détectée, notre système devait calculer la localisation précise de celle-ci en temps réel. À l'aide d'un prototype que nous avons développé et de divers équipements du laboratoire de robotique, nous avons pu démontrer la faisabilité de notre proposition.

Les essais de vérification que nous avons effectués ont démontré que notre système pouvait identifier plusieurs cibles de petite dimension avec une grande précision lorsqu'il se trouvait en position quasi stationnaire. Nos essais ont aussi démontré que nous

pouvions facilement détecter une cible lorsque notre appareil se déplaçait plus rapidement, cependant la précision de localisation diminue plus la vitesse augmente.

Bien que certains détails restent à améliorer, notamment au niveau de la précision du calcul de la localisation des cibles lorsque notre système est en déplacement rapide, nous croyons que notre concept possède de nombreux avantages. L'automatisation de processus tel que la détection de cible et le calcul de leurs positions en temps réel permettent de réduire les erreurs dues à une intervention humaine, d'accélérer le traitement des données et d'améliorer la sécurité des opérations. Les principales faiblesses de notre projet sont essentiellement liées aux contraintes imposées par la nécessité de développer notre projet à l'intérieur et ne serait probablement pas reproduit lors d'un développement à grande échelle dans un contexte réel.

Annexe A – Code Matlab

```
% Optitrack Matlab / NatNet Sample
%
%  Requirements:
%   - OptiTrack Motive 1.5 or later
%   - OptiTrack NatNet 2.5 or later
%   - Matlab R2013
%

function NatNetMatlabSample()
    delaybuffer = 150;
    delete(instrfind);
    XBEE = serial('COM3','BaudRate',115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, 'FlowControl',
    'none')
    fopen(XBEE)
    display('NatNet SIROIS_GAGNON Begin')

    global frameRate;
    lastFrameTime = -1.0;
    lastFrameID = -1.0;
    usePollingLoop = true;        % approach 1 : poll for mocap data in a tight loop using GetLastFrameOfData
    usePollingTimer = false;      % approach 2 : poll using a Matlab timer callback ( better for UI based apps )
    useFrameReadyEvent = false;   % approach 3 : use event callback from NatNet (no polling)
    useUI = true;

    persistent arr;
    % Open figure
    if(useUI)
        hFigure = figure('Name','OptiTrack NatNet Matlab Sample','NumberTitle','off');
    end

    try
        % Add NatNet .NET assembly so that Matlab can access its methods, delegates, etc.
        % Note : The NatNetML.DLL assembly depends on NatNet.dll, so make sure they
        % are both in the same folder and/or path if you move them.
        display('[NatNet] Creating Client.')
        curDir = pwd;
```

```
36.          mainDir = fileparts(fileparts(curDir));
37.          dllPath = fullfile('C:\Users\s26718\Desktop\NatNetSDK','lib','x64','NatNetML.dll');
38.          assemblyInfo = NET.addAssembly(dllPath);
39.
40.          % Create an instance of a NatNet client
41.          theClient = NatNetML.NatNetClientML(1); % Input = iConnectionType: 0 = Multicast, 1 = Unicast
42.          version = theClient.NatNetVersion();
43.          fprintf( '[NatNet] Client Version : %d.%d.%d.%d\n', version(1), version(2), version(3), version(4) );
44.
45.          % Connect to an OptiTrack server (e.g. Motive)
46.          display('[NatNet] Connecting to OptiTrack Server.')
47.          hst = java.net.InetAddress.getLocalHost;
48.          HostIP1 = char(hst.getHostAddress);
49.          disp(HostIP1)
50.          HostIP = char('192.168.1.100');
51.          flg = theClient.Initialize(HostIP1, HostIP); % Flg = returnCode: 0 = Success
52.          if (flg == 0)
53.              display('[NatNet] Initialization Succeeded')
54.          else
55.              display('[NatNet] Initialization Failed')
56.              disp(flg)
57.          end
58.
59.          % print out a list of the active tracking Models in Motive
60.          GetDataDescriptions(theClient)
61.
62.          % Test - send command/request to Motive
63.          [byteArray, retCode] = theClient.SendMessageAndWait('FrameRate');
64.          if(retCode ==0)
65.              byteArray = uint8(byteArray);
66.              frameRate = typecast(byteArray,'single');
67.          end
68.
69.          % get the mocap data
70.          if(usePollingTimer)
71.              % approach 2 : poll using a Matlab timer callback ( better for UI based apps )
72.              framePerSecond = 30;   % timer frequency
73.              TimerData = timer('TimerFcn',{@TimerCallback,theClient},'Period',1/framePerSecond,'ExecutionMode',
                 'fixedRate','BusyMode','drop');
74.              start(TimerData);
```

```
75.             % wait until figure is closed
76.             uiwait(hFigure);
77.         else
78.             if(usePollingLoop)
79.                 % approach 1 : get data by polling - just grab 5 secs worth of data in a tight loop
80.                 for idx = 1 : 10000
81.                    % Note: sleep() accepts [mSecs] duration, but does not process any events.
82.                    % pause() processes events, but resolution on windows can be at worst 15 msecs
83.                    java.lang.Thread.sleep(delaybuffer);
84.
85.                     % Poll for latest frame instead of using event callback
86.                     data = theClient.GetLastFrameOfData();
87.                     data123 = theClient.GetLastFrameOfData();
88.                     frameTime = data.fLatency;
89.                     frameID = data.iFrame;
90.
91.
92.                     % start of program
93.
94.
95.                     POSx = data.RigidBodies(1).x;
96.                     POSy = data.RigidBodies(1).y;
97.                     POSz = data.RigidBodies(1).z;
98.                     q = quaternion(data.RigidBodies(1).qy, data.RigidBodies(1).qx, data.RigidBodies(1).qz,
                                       data.RigidBodies(1).qw);
99.                     qRot =   quaternion( 0, 0, 0, 1);     % rotate pitch 180 to avoid 180/-180 flip for nicer
                            graphing
100.                    q = mtimes(q, qRot);
101.                    angles = EulerAngles(q,'zyx');
102.                    angleYaw = -angles(1) * 180.0 / pi;   % must invert due to 180 flip above
103.                    angleRoll = angles(2) * 180.0 / pi;
104.                    anglePitch = -angles(3) * 180.0 / pi;
105.                    datatosend = [123.456, 1000*POSx, 1000*POSy, 1000*POSz, anglePitch, angleYaw, angleRoll,
                            idx]
106.                    %datatosend1 = [1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8]
107.
108.                    fwrite(XBEE, 'DEBUT')
109.                    fwrite(XBEE, ' ')
110.                    LengthString = 0;
111.                    Extra_Character = 9;
```

```
112.                    for i = 1:length(datatosend)
113.                      datatosend_conv{i} = num2str(datatosend(i), '%18.8f');
114.                      LengthString = LengthString + length(num2str(datatosend(i), '%18.8f'));
115.                    end
116.                    if LengthString ~= 100
117.                        disp(LengthString)
118.                        datatosend_conv{Extra_Character}(1:(100-LengthString)) = '.';
119.                    end
120.
121.                    for j = 1:length(datatosend_conv)
122.                        fwrite(XBEE, datatosend_conv{j})
123.                        fwrite(XBEE, ' ')
124.                    end
125.                    fwrite(XBEE, 'FIN')
126.                    fwrite(XBEE, ' ')
127.                    java.lang.Thread.sleep(delaybuffer);
128.                    fwrite(XBEE, hex2dec('0A'), 'uint8')
129.                    LengthString1 = 0;
130.                    for m = 1:9
131.                      LengthString1 =  LengthString1 + length(datatosend_conv{m});
132.                    end
133.                    %disp(length(datatosend_conv{Extra_Character}))
134.                    datatosend_conv{Extra_Character} = [];
135.                    %disp(LengthString1)
136.
137.
138.
139.                    java.lang.Thread.sleep(delaybuffer);
140.
141.                    %eol = uint8([10])
142.                    %fwrite(XBEE, eol, 'char')
143.
144.
145.
146.
147.                    if(frameTime ~= lastFrameTime)
148.                        fprintf('FrameTime: %0.3f\tFrameID: %5d\n',frameTime, frameID);
149.                        lastFrameTime = frameTime;
150.                        lastFrameID = frameID;
151.                    else
```

```
                            display('Duplicate frame');
                        end
                    end
                else
                    % approach 3 : get data by event handler (no polling)
                    % Add NatNet FrameReady event handler
                    ls = addlistener(theClient,'OnFrameReady2',@(src,event)FrameReadyCallback(src,event));
                    display('[NatNet] FrameReady Listener added.');
                    % wait until figure is closed
                    uiwait(hFigure);
                end
            end

        catch err
            display(err);
        end

        % cleanup
        if(usePollingTimer)
            stop(TimerData);
            delete(TimerData);
        end
        theClient.Uninitialize();
        if(useFrameReadyEvent)
            if(~isempty(ls))
                delete(ls);
            end
        end
        clear functions;

        display('NatNet Sample End')

    end

    % Test : process data in a Matlab Timer callback
    function TimerCallback(obj, event, theClient)

        frameOfData = theClient.GetLastFrameOfData();
        UpdateUI( frameOfData );

```

```
192.  end
193.
194.  % Test : Process data in a NatNet FrameReady Event listener callback
195.  function FrameReadyCallback(src, event)
196.
197.      frameOfData = event.data;
198.      UpdateUI( frameOfData );
199.
200.  end
201.
202.  % Update a Matlab Plot with values from a single frame of mocap data
203.  function UpdateUI( frameOfData )
204.
205.      persistent lastFrameTime;
206.      persistent lastFrameID;
207.      persistent hX;
208.      persistent hY;
209.      persistent hZ;
210.      persistent arrayIndex;
211.      persistent frameVals;
212.      persistent xVals;
213.      persistent yVals;
214.      persistent zVals;
215.      persistent bufferModulo;
216.
217.      global frameRate;
218.
219.      % first time - generate an array and a plot
220.      if isempty(hX)
221.          % initialize statics
222.          bufferModulo = 256;
223.          frameVals = 1:255;
224.          xVals = zeros([1,255]);
225.          yVals = zeros([1,255]);
226.          zVals = zeros([1,255]);
227.          arrayIndex = 1;
228.          lastFrameTime = frameOfData.fLatency;
229.          lastFrameID = frameOfData.iFrame;
230.
231.          % create plot
```

```
232.          hX = plot(frameVals, xVals, 'color', 'r');
233.          hold on;
234.          hY = plot(frameVals, yVals, 'color', 'g');
235.          hZ = plot(frameVals, zVals, 'color', 'b');
236.          title('Mocap Angle Plot');
237.          xlabel('Frame number');
238.          ylabel('Angle (degrees)');
239.          set(gca,'YLim',[-180 180]);
240.          set(gca,'XGrid','on','YGrid','on');
241.      end
242.
243.      % calculate the frame increment based on mocap frame's timestamp
244.      % in general this should be monotonically increasing according
245.      % To the mocap framerate, however frames are not guaranteed delivery
246.      % so to be accurate we test and report frame drop or duplication
247.      newFrame = true;
248.      droppedFrames = false;
249.      frameTime = frameOfData.fLatency;
250.      frameID = frameOfData.iFrame;
251.      calcFrameInc = round( (frameTime - lastFrameTime) * frameRate );
252.      % clamp it to a circular buffer of 255 frames
253.      arrayIndex = mod(arrayIndex + calcFrameInc, bufferModulo);
254.      if(arrayIndex==0)
255.          arrayIndex = 1;
256.      end
257.      if(calcFrameInc > 1)
258.          % debug
259.          % fprintf('\nDropped Frame(s) : %d\n\tLastTime : %.3f\n\tThisTime : %.3f\n', calcFrameInc-1,
                  lastFrameTime,                          frameTime);
260.          droppedFrames = true;
261.      elseif(calcFrameInc == 0)
262.          % debug
263.          % display('Duplicate Frame')
264.          newFrame = false;
265.      end
266.
267.      % debug
268.      % fprintf('FrameTime: %0.3f\tFrameID: %d\n',frameTime, frameID);
269.
270.      try
```

```
271.          if(newFrame)
272.              if(frameOfData.RigidBodies.Length() > 0)
273.
274.                  rigidBodyData = frameOfData.RigidBodies(1);
275.
276.                  % Test : Marker Y Position Data
277.                  % angleY = data.LabeledMarkers(1).y;
278.                  % Test : Rigid Body Y Position Data
279.                  % angleY = rigidBodyData.y;
280.
281.                  % Test : Rigid Body 'Yaw'
282.                  % Note : Motive display euler's is X (Pitch), Y (Yaw), Z (Roll), Right-Handed (RHS), Relative
      Axes
283.                  % so we decode eulers heres to match that.
284.                  q = quaternion( rigidBodyData.qx, rigidBodyData.qy, rigidBodyData.qz, rigidBodyData.qw );
285.                  qRot = quaternion( 0, 0, 0, 1);     % rotate pitch 180 to avoid 180/-180 flip for nicer graphing
286.                  q = mtimes(q, qRot);
287.                  angles = EulerAngles(q,'zyx');
288.                  angleX = -angles(1) * 180.0 / pi;   % must invert due to 180 flip above
289.                  angleY = angles(2) * 180.0 / pi;
290.                  angleZ = -angles(3) * 180.0 / pi;   % must invert due to 180 flip above
291.
292.                  if(droppedFrames)
293.                      for i = 1 : calcFrameInc
294.                          fillIndex = arrayIndex - i;
295.                          if(fillIndex < 1)
296.                              fillIndex = bufferModulo-(abs(fillIndex)+1);
297.                          end
298.                          xVals(fillIndex) = angleX;
299.                          yVals(fillIndex) = angleY;
300.                          zVals(fillIndex) = angleZ;
301.                      end
302.                  end
303.
304.                  % update the array/plot for this frame
305.                  xVals(arrayIndex) = angleX;
306.                  yVals(arrayIndex) = angleY;
307.                  zVals(arrayIndex) = angleZ;
308.                  set(hX, 'YData', xVals);
309.                  set(hY, 'YData', yVals);
```

```
                set(hZ, 'YData', zVals);

            end
        end
    catch err
        display(err);
    end

    lastFrameTime = frameTime;
    lastFrameID = frameID;

end

% Print out a description of actively tracked models from Motive
function GetDataDescriptions( theClient )

    dataDescriptions = theClient.GetDataDescriptions();

    % print out
    fprintf('[NatNet] Tracking Models : %d\n\n', dataDescriptions.Count);
    for idx = 1 : dataDescriptions.Count
        descriptor = dataDescriptions.Item(idx-1);
        if(descriptor.type == 0)
            fprintf('\tMarkerSet \t: ');
        elseif(descriptor.type == 1)
            fprintf('\tRigid Body \t: ');
        elseif(descriptor.type == 2)
            fprintf('\tSkeleton \t: ');
        else
            fprintf('\tUnknown data type : ');
        end
        fprintf('%s\n', char(descriptor.Name));
    end

    for idx = 1 : dataDescriptions.Count
        descriptor = dataDescriptions.Item(idx-1);
        if(descriptor.type == 0)
            fprintf('\n\tMarkerset : %s\t(%d markers)\n', char(descriptor.Name), descriptor.nMarkers);
            markerNames = descriptor.MarkerNames;
            for markerIndex = 1 : descriptor.nMarkers
```

```
350.                 name = markerNames(markerIndex);
351.                 fprintf('\t\tMarker : %-20s\t(ID=%d)\n', char(name), markerIndex);
352.             end
353.         elseif(descriptor.type == 1)
354.             fprintf('\n\tRigid Body : %s\t\t(ID=%d, ParentID=%d)\n', char(descriptor.Name),descriptor.ID,de-
    scriptor.parentID);
355.         elseif(descriptor.type == 2)
356.             fprintf('\n\tSkeleton : %s\t(%d bones)\n', char(descriptor.Name), descriptor.nRigidBodies);
357.             %fprintf('\t\tID : %d\n', descriptor.ID);
358.             rigidBodies = descriptor.RigidBodies;
359.             for boneIndex = 1 : descriptor.nRigidBodies
360.                 rigidBody = rigidBodies(boneIndex);
361.                 fprintf('\t\tBone : %-20s\t(ID=%d, ParentID=%d)\n', char(rigidBody.Name), rigidBody.ID, rigid-
    Body.parentID);
362.             end
363.         end
364.     end
365. end
```

Annexe B – Code Python

```
 1.  import io
 2.  import time
 3.  import picamera
 4.  import cv2
 5.  import numpy as np
 6.  import sys
 7.  import cmath
 8.  from scipy import linalg
 9.  import serial
10.  import math
11.  import os
12.  #create video capture
13.  cap = cv2.VideoCapture(0)
14.  cap.set(cv2.CAP_PROP_FPS, 8)
15.  #initialisation de la matrice d'élévation
16.  temps_precedent = 0
17.  Elevation = [ [ 0 for i in range(650) ] for j in range(650) ]
18.  XOFFSET = 325
19.  YOFFSET = 325
20.
21.  for i in range (-40+XOFFSET,40+XOFFSET):
22.       for j in range (150+YOFFSET, 210+YOFFSET):
23.                 Elevation[i][j] = 75.356
24.  for i in range (-20 + XOFFSET, 25 + XOFFSET):
25.       for j in range (55+ YOFFSET, 100 + YOFFSET):
26.             Elevation[i][j] = 11.684
27.  for i in range (-20 + XOFFSET, 31 + XOFFSET):
28.       for j in range (-20+YOFFSET, 22 +YOFFSET):
29.             Elevation[i][i] = 15.5
30.  for i in range (-120 + XOFFSET, -90+XOFFSET):
31.       for j in range (120 + YOFFSET, 180 + YOFFSET):
32.             Elevation[i][j] = 32
33.  for i in range (-150 + XOFFSET, -90 + XOFFSET):
34.       for j in range (48+YOFFSET, 100 +YOFFSET):
35.             Elevation[i][j] = 68.98
36.  for i in range (-170+XOFFSET, -135+XOFFSET):
```

```
37.         for j in range (175+YOFFSET, 240+YOFFSET):
38.               Elevation[i][j] = 68.98
39.
40.   print('FINI INITIALISATION CARTE')
41.   #initialisation of the global variables
42.   ser = serial.Serial("/dev/XBEE", 115200)
43.
44.   matrice_transfo_static = np.array([[1, 0, 0, 4.1],[0 , 1, 0, -4.3], [0, 0, 1, -      5.6], [0, 0, 0, 1]])
45.   #test1 = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 1000], [0, 0, 0, 1]])
46.   K_Mat = np.array([[634.782, 0, 320.8814], [0, 636.25833, 240.87371], [0, 0,    1]], dtype = np.float32)
47.   #K_inv = linalg.inv(K_Mat)
48.   kc = np.array([0.0974, -0.3001, 0.00074, -0.00030, 0.0000], dtype = np.float32)
49.   Matrice_Test = np.array([[0.70711, 0, 0.70711, 0],[0, 1, 0, 0], [-0.70711, 0,  0.70711, 1000],[0, 0, 0, 1]])
50.
51.   #matrice_transfo = np.dot(test1, matrice_transfo_static)
52.
53.
54.   #if you want to debug and test without optitrack
55.   OPTITRACK_OVERRIDE = 0
56.   EXIT_FLAG = 0
57.   XSCALING = 12.25
58.   YSCALING = 12.458
59.   ZSCALING = 12.5
60.
61.   XTARGETOFFSET = 6
62.   YTARGETOFFSET = 5.5
63.
64.   PositionCibleX = 0
65.   PositionCibleY = 0
66.
67.
68.   print('Debut du programme')
69.   while(1):
70.       #read the frames
71.         if OPTITRACK_OVERRIDE == 0:
72.         #INSERT XBEE READING HERE
73.               while (1) :
74.                     try:
75.                           # _, frame = cap.read()
76.                           rawData = ser.readline()
```

```
                        parsedData = rawData.split(" ")
                        if parsedData[0] == 'DEBUT':
                              if len(parsedData) > 10 :
                                    if parsedData[10] == 'FIN':
                                          #print(parsedData)
                                          break
                                    else:
                                          print ('Next String Corrupt1')
                              else :
                                    print ('Next String Corrupt2')
                        else :
                              print('Next String Corrupt3')

                  except KeyboardInterrupt :
                        break
                        EXIT_FLAG = 1


      if EXIT_FLAG == 1:
            break
      ser.flushInput()
      if OPTITRACK_OVERRIDE == 0:
            POSITIONX = float(parsedData[2])/XSCALING
            POSITIONZ = float(parsedData[3])/ZSCALING
            POSITIONY = -1 * float(parsedData[4])/YSCALING
            ROLL = float(parsedData[5])*3.141596/180
            YAW = -1*float(parsedData[6])*3.141596/180
            PITCH = -1*float(parsedData[7])*3.141596/180
            ID = float(parsedData[8])
            ID = int(ID)
            #print ID
            ca = math.cos(YAW)
            sa = math.sin(YAW)
            cb = math.cos(ROLL)
            sb = math.sin(ROLL)
            cg = math.cos(PITCH)
            sg = math.sin(PITCH)

      #FINISH IT HERE
            Mat_PITCH = np.array([[1,0,0,0],[0,cg,sg,0],[0,-sg,cg,0],[0,0,0,1]])
```

```
117.              Mat_YAW = np.array([[ca,sa,0,0],[-sa,ca,0,0],[0,0,1,0],[0,0,0,1]])
118.              Mat_ROLL = np.array([[cb,0,-sb,0],[0,1,0,0],[sb,0,cb,0],[0,0,0,1]])
119.              Mat_TRANSLATION = np.array([[1,0,0,POSITIONX],[0,1,0,POSITIONY],[0,0,1,POSITIONZ],[0,0,0,1]])
120.              matrice_transfo_optitrack = np.dot(Mat_PITCH, np.dot(Mat_YAW, np.dot(Mat_ROLL, Mat_TRANSLATION)))
121.              MAT_TRANSFO = np.dot(matrice_transfo_static, matrice_transfo_optitrack)
122.              #print MAT_TRANSFO
123.        #getting a more recent frame
124.        for i in range(0,6):
125.              cap.grab()
126.
127.
128.        _,frame = cap.read()
129.
130.      #red thresholding
131.        hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
132.        lowerHUE = cv2.inRange(hsv,np.array((0, 50, 100)), np.array((10, 255, 255)))
133.        upperHUE = cv2.inRange(hsv,np.array((160, 50, 100)), np.array((179, 255, 255)))
134.        redThresh = cv2.addWeighted(lowerHUE, 1.0, upperHUE, 1.0, 1.0, 0.0)
135.
136.      #filtering edges to remove as much noise as possible
137.        redThresh = cv2.blur(redThresh,(2,2))
138.        thresh2 = redThresh.copy()
139.
140.      #setting params for keypoint detection
141.        params = cv2.SimpleBlobDetector_Params()
142.        params.minThreshold = 240
143.        params.maxThreshold = 255
144.        params.filterByArea = False
145.        params.filterByCircularity = False
146.        params.filterByConvexity = False
147.        params.filterByColor = True
148.        params.blobColor = 255
149.
150.      #simpleBlob_detector to detect keypoints
151.        detector = cv2.SimpleBlobDetector_create(params)
152.        keypoints = detector.detect(thresh2)
153.        inputPoints = np.zeros((len(keypoints), 1, 2), dtype=np.float32)
154.
155.        #putting circles on center of blobs
156.        for i in range(len(keypoints)):
```

```
157.                cv2.circle(frame,(int(keypoints[i].pt[0]),int(keypoints[i].pt[1])),5,255,-1)
158.                inputPoints[i,0,0] = keypoints[i].pt[0]
159.                inputPoints[i,0,1] = keypoints[i].pt[1]
160.
161.          #if there is a target....
162.
163.
164.          if len(keypoints) > 0:
165.
166.                #use the center of blobs and undistort it to get normalized pixels (0 -> 1 for z = 1)
167.                outputPoints = cv2.undistortPoints(inputPoints, K_Mat, kc)
168.
169.                #calculate the origin (center of the camera in the OptiTrack Reference frame)
170.                origine = np.dot(matrice_transfo_static, np.array([[0], [0], [0], [1]]))
171.                origine = np.dot(Mat_YAW, origine)
172.                origine = np.dot(Mat_PITCH, origine)
173.                origine = np.dot(Mat_ROLL, origine)
174.                origine = np.dot(Mat_TRANSLATION, origine)
175.    #           print(origine)
176.                os.system('clear')
177.                #for each target..
178.                for i in range(len(keypoints)):
179.                      outputPoints[i,0,1] = (-1)*outputPoints[i,0,1]
180.
181.                      #create a vector and a walking point
182.                      cibles = np.array([[outputPoints[i,0,0]],[outputPoints[i,0,1]], [-1], [1]])
183.                      vecteurcible = np.dot(matrice_transfo_static, cibles)
184.                      vecteurcible = np.dot(Mat_YAW, vecteurcible)
185.                      vecteurcible = np.dot(Mat_PITCH, vecteurcible)
186.                      vecteurcible = np.dot(Mat_ROLL, vecteurcible)
187.                      vecteurcible = np.dot(Mat_TRANSLATION, vecteurcible)
188.
189.
190.                      position = origine
191.                      vecteur_direction = vecteurcible-origine
192.                      #walk along the vector until you hit the digital élévation mat
193.                      while(position[2,0] >  Elevation[int(position[0,0]+XOFFSET)][int(position[1,0]+YOFFSET)]) :
194.                            position = position + vecteur_direction
195.                            if (position[0,0]+XOFFSET)>640 or (position[0,0]+XOFFSET)<10 or (position[1,0] +
196.                            YOFFSET) > 640 or (position[1,0] + YOFFSET)<10:
```

```
197.                                    print('out of bounds')
198.                                    break
199.
200.                        #print the x and y coordinates of the exact location on the digital map/OptiTrack reference
201.                        Diff = math.sqrt((PositionCibleX - position[0,0])*(PositionCibleX - position[0,0]) +
202.                               (PositionCibleY - position[1,0])*(PositionCibleY-position[1,0]))
203.                        POS = 'Cible Trouver !! Position X :' + repr(position[0,0]+XTARGETOFFSET) + '    Position Y
                           :' + repr(position[1,0]+YTARGETOFFSET) + '     Elevation:' +
                           repr(Elevation[int(position[0,0]+XOFFSET)][int(position[1,0]+YOFFSET)])
204.                        #print('difference')
205.                        #print(Diff)
206.                        #print cap.get(cv2.CAP_PROP_FPS)
207.                        print(POS)
208.    #     temps = time.time()
209.    #     print((temps-temps_precedent))
210.    #     temps_precedent = temps
211.    #     print('YAW:' + repr(YAW))
212.    #     print('ROLL:' + repr(ROLL))
213.    #     print('PITCH:' + repr(PITCH))
214.    #     print('X:' + repr(POSITIONX))
215.    #     print('Y:' + repr(POSITIONY))
216.    #     print('Z:' + repr(POSITIONZ))
217.    #     print('-------------------------------------------------------------------------------------------------------
           ---------------END')
218.
219.         # Show it, if key pressed is 'Esc', exit the loop
220.          #print('saving DATA at shot_test_'+repr(ID)+'.png')
221.          #cv2.imwrite('shot_test_'+repr(ID)+'.png',frame)
222.          cv2.imshow('frame',frame)
223.          cv2.imshow('thresh',thresh2)
224.          if cv2.waitKey(33)== 27:
225.                break
226.    # Clean up everything before leaving
227.    cv2.destroyAllWindows()
228.    cap.release()
229.    ser.close()
```