

Projet ModRADAR

Rapport de mi-parcours

Mohammed Al-Ghamdi

mohammed.al_ghamdi@ensta-bretagne.org

Hendrik Hiddinga

hendrik.hiddinga@ensta-bretagne.org

Chadi Jadraque

chadi.jadraque@ensta-bretagne.org

Alexis Motet

alexis.motet@ensta-bretagne.org

Madeleine Polycarpe

madeleine.polycarpe@ensta-bretagne.org

Résumé

ModRADAR est un projet qui vise à caractériser et à évaluer la facilité d'utilisation des radars uRAD. En effet, les radars de petites tailles se sont démocratisés de manière récente, et sont désormais abordables pour les écoles d'ingénieurs afin de les intégrer aux divers projets de leurs élèves. Afin de déterminer si les radars uRAD sont pertinents pour cet usage précis, nous cherchions à les utiliser dans un système de suivi des coureurs sur une piste d'athlétisme. Nous souhaitions connaître la vitesse et la position de ces derniers au cours de la course et envisagions de les mettre en réseau pour couvrir la plus grande surface possible. Nous avons réalisé notre propre interface graphique afin qu'elle s'adapte à nos besoins (simplicité d'utilisation notamment). Nous avons utilisé le mode de modulation fréquentielle de ces radars permettant de mesurer distance et vitesse. Pour avoir une bonne résolution sur les vitesses mesurées, nous sommes partis des données brutes et nous les avons traiter avec nos algorithmes. Nous avons aussi créé un système de cibles pour associer les distances à des coureurs. Nous sommes finalement parvenus à obtenir de bons résultats en testant notre système en condition réelle sur de vrais coureurs.

Mots clés

Rapport de projet, radar, ondes électromagnétiques.

Abstract

ModRADAR is a project that aims to characterise and evaluate the usability of uRAD radar cards. Indeed, small-scale radars have recently become popular and affordable for engineering schools to integrate into their students' various projects. In order to determine whether this particular radar is relevant for this particular purpose, we are looking to use it in a system to track runners on an athletics track. We want to know the speed and position of the runners during the race and plan to connect different cards to cover the largest possible area. So far, we have made our own graphical interface to fit our needs. We started with the raw data and processed it with our algorithms, using the card's adapted modes to measure speed and distance. We finally created a target system to associate distances with runners. We will now be able to focus on connecting several cards and creating an interface that presents the conclusions obtained by combining the information from each map.

Keywords

Project report, radar, electromagnetic radiations.

Remerciements

Nous tenons à remercier le Dr Fabrice COMBLET, le Dr Abdelmalek TOUMI, M. Didier TANGUY, le Dr Fabrice LE BARS et le Dr Jean-Christophe CEXUS pour leur

accompagnement dans les domaines scientifiques, ainsi que M. Joël CHAMPEAU pour son aide à la gestion de l'équipe en mode agile.

Table des matières

1 Situation au début de la seconde partie du projet	2
1.1 Travail accompli durant la première partie	2
1.2 Objectifs initiaux au début de la seconde partie	2
2 Gestion du projet	3
2.1 Notre organisation	3
2.2 Notre avancement et notre diagramme de Gantt	3
3 Architecture fonctionnelle et physique	5
3.1 Liste des exigences et identification des fonctions principales	5
3.1.1 Tableau des exigences	5
3.1.2 Identification des fonctions principales du système	5
3.2 Architecture fonctionnelle	6
3.3 Architecture physique	8
4 Développement	9
4.1 Introduction de la partie technique	9
4.1.1 Trilateration	9
4.1.2 Superposition	11
4.2 Réseau	12
4.2.1 Présentation du réseau	12
4.2.2 Fonctionnement du réseau	13
4.2.3 Latence et fréquence d'acquisition	14
4.3 Traitement du signal et cibles-fantômes	14
4.3.1 Bref retour sur le semestre 4	14
4.3.2 Associations des fréquences	14
4.3.3 Zero-padding et résolution en vitesse	16
4.4 Détection vidéo et suivi de cibles	17
4.4.1 La détection d'objets	17
4.4.2 Le suivi de cibles par filtre de Kalman	18
4.5 Interface graphique	18
4.5.1 Technologies utilisées	18
4.5.2 Présentation de l'interface graphique	19
4.6 Résultats et pistes d'amélioration	20
4.6.1 Présentation des résultats	20
4.6.2 Pistes d'amélioration	21
5 Tests réalisés	21
5.1 Tests unitaires	21
5.1.1 Détail du test unitaire pour notre algorithme à seuil adaptatif	22
5.1.2 Détail du test unitaire pour notre algorithme d'associations de fréquences	22
5.2 Tests d'intégration	22
5.2.1 Test d'intégration de la détection vidéo	23
5.2.2 Test d'intégration du suivi de cibles	23

5.3 Test de validation fonctionnelle	23
Annexes	25
A Annexe	25
A.1 Code du test unitaire de l'algorithme CA-CFAR (MATLAB)	25
A.2 Code du test unitaire de l'algorithme d'associations de fréquences (Python)	29
Liste des figures	33
Glossaire	34
Bibliographie	34

(

Introduction

Notre projet est centré autour de petits radars uRAD embarqués sur des cartes électroniques. Ces radars sont des radars à onde continue modulée en fréquence qui peuvent mesurer des distances supérieures à 50 mètres et des vitesses supérieures à 20m/s. Nous avons décidé au semestre 4 de les utiliser pour mesurer des vitesses de coureurs sur une piste, afin de créer un système utilisable par un professeur de sport ou un coach sportif et permettant de réaliser des "photos-finish".

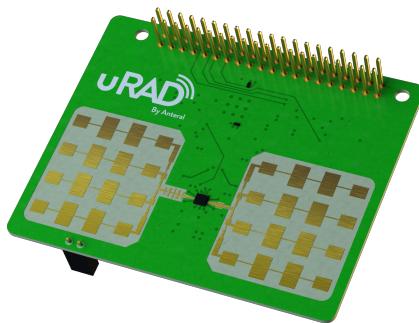


FIGURE 1 – Un radar uRAD

1 Situation au début de la seconde partie du projet

1.1 Travail accompli durant la première partie

Lors de la première partie, nous avons défini un premier cahier des charges et nous avons réalisé un état de l'art sur le traitement du signal et les radars à onde continue modulée en fréquence.

Puis nous sommes partis du signal brut renvoyé par un radar embarqué sur carte Arduino et avons appliqué sur ce signal tout le traitement du signal nécessaire pour arriver à des mesures de distance et de vitesse. Enfin, nous avons proposé une méthode de suivi de cibles naïve fondée sur l'algorithme Hongrois.

1.2 Objectifs initiaux au début de la seconde partie

Au début de la deuxième partie, nous avions en notre possession trois radars uRAD, un était porté sur Arduino et les deux autres sur Raspberry Pi 4.

Nous voulions alors les utiliser pour localiser des coureurs sur une piste. Cependant, nous étions bloqué par le fait que les radars ne renvoient pas d'angle, mais seulement des distances, c'est pourquoi nous avons commencé à imaginer un système de trilateration. Ce système nous a demandé de mettre en communication les cartes qui portaient les radars : nous pensions à l'origine tout faire avec un serveur Web, mais nous avons eu la chance de découvrir l'existence de Redis, un système de stockage très hautes performance, système qui s'est révélé parfaitement adapté à notre projet.

Au début du projet, nous n'avions pas pensé à ajouter une vidéo pour filmer la piste, et encore moins à ajouter les mesures des radars sur la vidéo. Ce n'est qu'après une phase expérimentale très dense que nous nous sommes orientés vers ce qui nous paraissait le plus adapté à notre cahier des charges.

2 Gestión del proyecto

2.1 Nuestra organización

Durante la segunda parte de este proyecto, nosotros habíamos planeado una reunión el martes 25 de enero de 2022 para nos dar una repartición de las tareas más o menos precisas a cada persona de nuestro grupo y escribir una esbozo de cahier des charges. Sin embargo, dado que nosotros no teníamos una línea directriz real, la reunión con los profesores el martes 2 de febrero nos permitió establecer un primer diagrama de Gantt en su lugar.

Como explicada en la sección, **1.2 Objectifs initiaux au début de la seconde partie**, nosotros hemos avanzado durante todo este semestre de proyecto en tener una gran margen de libertad y reformular nuestros objetivos todo en adaptándonos a nuestro cahier des charges. Las decisiones de crear una página Web en localhost para la interfaz gráfica, de utilizar una cámara y crear un red local a partir de un router Wifi-Lan, han sido efectuadas cuando encontramos una restricción con los radares o cuando hemos realizado que podíamos poner en marcha una solución complementaria y adicional a las características de las tarjetas Urads, particularmente la puesta en marcha de la interfaz gráfica.

Por otra parte, la detección de video ha sido así un elemento que parecía inherente a nuestro proyecto cuando lo creamos.

Un elemento esencial a nuestra avanzada fue la regularidad de los tests prácticos que hemos hecho. Esta regularidad, como veremos en nuestro diagrama de Gantt, nos ha sido de gran ventaja. Hemos podido tomar rápidamente muchas decisiones rápidamente tales como el abandono del sistema de triangulación, la puesta en punto de una interfaz gráfica y el uso de una cámara. Hemos podido proceder a constantes correcciones en nuestro sistema pero sobre todo esto nos ha evitado alejarnos de nuestro cahier des charges.

En cuanto a la coordinación, hemos continuado utilizando las metodologías del semestre anterior para nuestro equipo. De hecho, hemos puesto en marcha un diagrama de Gantt para efectuar los sprints. Sin embargo, los sprints a menudo eran demasiado cortos. Nuestra coordinación ha sido, en realidad, más guiada por los tests individuales de martes que hemos hecho y quién a la conclusión de estas sesiones, nos permitían continuar o modificar nuestros trabajos y investigaciones. Hemos continuado teniendo un diario de bordo al principio del semestre todo en comunicando a través de nuestra conversación Messenger que es realmente eficaz, y luego únicamente a través de nuestra conversación Messenger que era suficiente para la finalización del proyecto.

Para las partes técnicas, como la programación de la interfaz o los algoritmos de detección de video bajo OpenCv, hemos utilizado un repositorio GitHub. Este conjunto de medidas nos ha ayudado a mantener mejor la metodología Agile dentro de nuestro equipo.

2.2 Nuestro avance y nuestro diagrama de Gantt

Dentro de esta sección, vamos a presentar nuestro diagrama de Gantt final. Hay una repartición de las tareas que fue realizada durante toda la segunda parte de este proyecto.

Finalmente, los diferentes sprints y los diferentes tests de nuestro sistema en condición real han sido factores esenciales para nuestro avance.

GANTT project

Nom	Date de début	Date de fin
Etude des concepts d'ingénierie système	11/01/22	01/02/22
Planification des tâches du semestre(Chadi)	11/01/22	01/02/22
Test de la carte Arduino sur piste (Alexis)	11/01/22	01/02/22
Recherche sur la mise en Réseau de carte...	11/01/22	01/02/22
Recherche PeopleTracking(Madeleine)	11/01/22	01/02/22
Réunion début de semestre(Tous)	25/01/22	25/01/22
Reunion avec les prof	02/02/22	02/02/22
Mise en réseau des cartes	02/02/22	18/02/22
Recherche Machine Learning(Madeleine)	02/02/22	18/02/22
Recherche pour utiliser Kalman(Hendrik)	02/02/22	18/02/22
Test carte Rasperry (Chadi)	02/02/22	18/02/22
Ingenierie Système(Mohamed)	02/02/22	18/02/22
Test 2 radars (Tous)	08/02/22	08/02/22
Site Web (Alexis)	15/02/22	25/03/22
Test de 3 radars gymnase (Tous)	22/02/22	22/02/22
Traitement Image (Chadi)	22/02/22	11/03/22
Detection Video(Madeleine)	22/02/22	11/03/22
Application Kalman	22/02/22	11/03/22
Recherche Réseau(Mohamed)	22/02/22	11/03/22
Test Gymnase	01/03/22	01/03/22
Test piste(Tous)	15/03/22	15/03/22
Prise en main Interface graphique(Chadi)	15/03/22	01/04/22
Essai Detection Vidéo(Madeleine)	15/03/22	01/04/22
Ingenierie Système(Mohamed)	15/03/22	01/04/22
Essai et Test Kalman(Hendrik)	15/03/22	01/04/22
Test piste(Tous)	22/03/22	22/03/22
Amélioration Site (Alexis)	28/03/22	01/04/22
Début de la Redaction(Tous)	04/04/22	11/04/22
Dernier Test et Réunion (Tous)	05/04/22	05/04/22

FIGURE 2 – Tâche donnée à chaque personne du groupe

Le diagramme de Gantt ci-dessous, constitue le fil conducteur de la démarche que nous avons employée. Nous avons gardé un code couleur différent pour chaque membre de l'équipe et mis en noir, les réunions et travaux où tout le monde était présent.

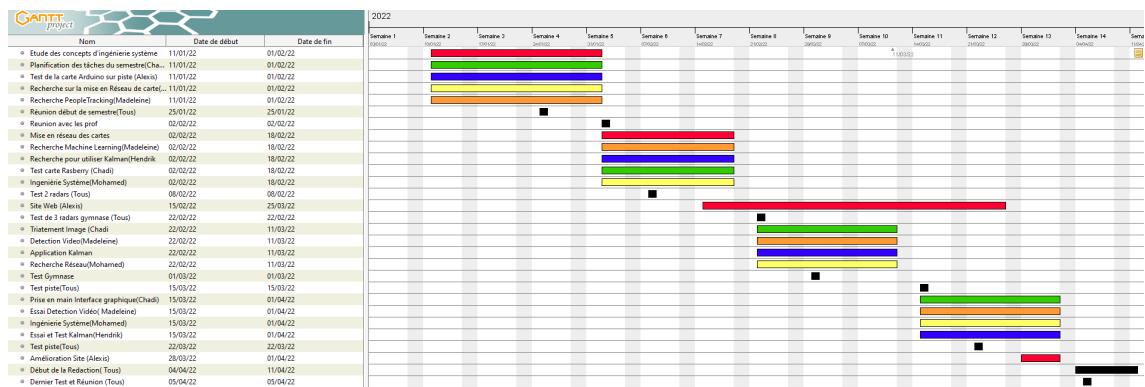


FIGURE 3 – Diagramme de Gantt du S4

3 Architecture fonctionnelle et physique

3.1 Liste des exigences et identification des fonctions principales

3.1.1 Tableau des exigences

Les exigences du système se divisent entre les fonctions et les contraintes: Nous avons isolé les suivantes :

Ex	Nom	Description	Type
E01	PoisDimMax	Le système doit être transportable	Contrainte
E02	MontagePossible	Le système doit être facilement montable et démontable	Contrainte
E03	MesureVit	Le système doit pouvoir renvoyer la vitesse d'un coureur et sa distance par rapport aux radars	Fonction
E04	VideoLive	Le système doit pouvoir afficher la vidéo avec les informations de vitesse et de distance des coureurs	Contrainte
E05	PlusieursPers	Le système doit pouvoir détecter plusieurs coureurs	Fonction
E06	PlusieursVit	Le système doit pouvoir mesurer plusieurs vitesses en même temps	Fonction
E07	Serveur	Le système doit stocker les mesures sur un serveur web local	Fonction
E08	RaspberryAcqui	Le système doit acquérir et traiter rapidement les signaux reçus	Fonction
E09	Réseaux	Le système doit pouvoir être mis en réseau	Fonction
E10	Tracker	Le système doit pouvoir identifier les coureurs en temps réel sur la vidéo	Fonction
E11	PageWeb	Le système doit posséder une page web reliée à une base de donnée et un flux vidéo	Fonction
E12	Filtre	Le système doit pouvoir utiliser des filtres adaptés	Fonction
E13	ServeurVit	Le système doit récupérer rapidement les données dans sa base de données et les renvoyer aux clients	Fonction
E14	CameraLag	Le système doit ne pas avoir de retard entre l'enregistrement et l'affichage	Contrainte

TABLE 1 – Tableau des Exigences

3.1.2 Identification des fonctions principales du système

Alors qu'au semestre précédent, on souhaitait que le système puisse être capable de détecter un, voire plusieurs coureurs, sur une piste et mesurer leur vitesse ainsi que leur position par rapport aux radars, nous avons au fur et à mesure crée un système faisant intervenir trois radars, une interface et une caméra connectés via un routeur WIFI, dont le déploiement est expliqué dans la partie suivante. Nous avons alors, durant ce semestre choisi d'orienter notre projet différemment. L'utilisation des nouveaux outils comme la base de données Redis ou l'utilisation de la détection vidéo sous OpenCV, nous a

ainsi amené à mettre à jour nos fonctions principales et plus tard nous le verrons nos architectures physique et fonctionnelle.

Ainsi, nous avons définis 5 fonctions principales du système, :

- **FP1** : Recevoir les données brutes du coureur
- **FP2** : Traiter correctement le signal reçu
- **FP3** : Sauvegarder les données vitesse/distance
- **FP4** : Acquérir les données visuelles de la course
- **FP5** : Associer le couple distance/vitesse au coureur

3.2 Architecture fonctionnelle

La fonction principale de notre projet était de mesurer la vitesse des coureurs comme au semestre précédent. On présente ci-dessous, le diagramme de tête à corne qui permet de répondre aux questions sur quoi la fonction agit-elle, à qui rend-elle service et pour quoi cette fonction existe-t-elle et reste ainsi similaire à celle élaborée au semestre précédent.

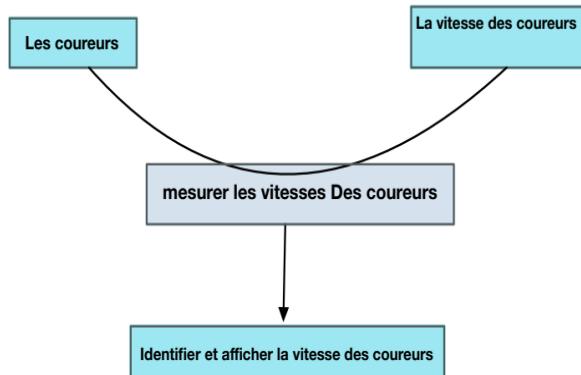


FIGURE 4 – Diagramme de tête à corne du S4

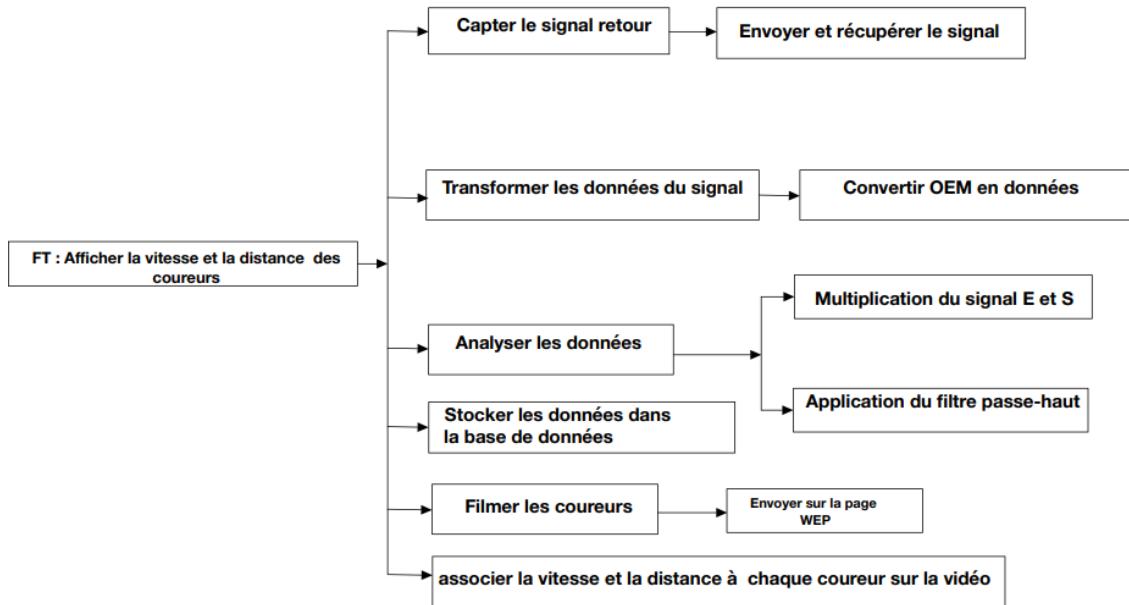


FIGURE 5 – Diagramme FAST du S4

Sur cette figure ci-dessus, nous avons réalisé notre diagramme FAST. Son utilité est de donner une décomposition de notre fonction finale en plusieurs fonctions techniques pour nous permettre de comprendre comment et quand la fonction finale sera exécutée. On a ainsi une vue d'ensemble de l'exécution de cette fonction. Elle a ainsi été mise à jour par rapport au diagramme FAST du semestre 3, avec notamment une nouvelle décomposition faisant intervenir le stockage des données et la restitution via l'interface graphique.

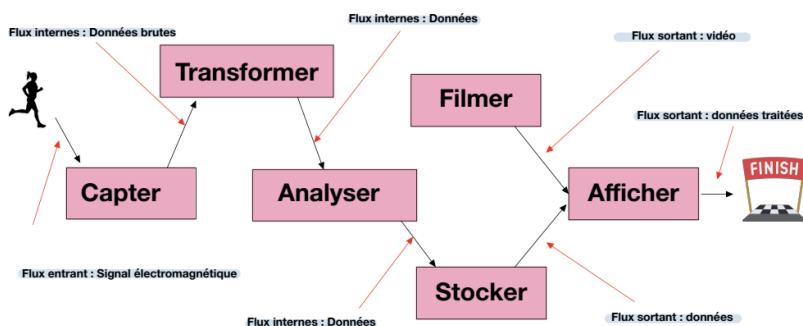


FIGURE 6 – Architecture fonctionnelle du S4

L'architecture fonctionnelle présentée ci-dessus, a pour objectif d'identifier les flux internes de notre système mais aussi les liens entre les fonctions et l'environnement. Cette architecture découle du diagramme FAST réalisé précédemment et est alors complétée

par les flux internes.

3.3 Architecture physique

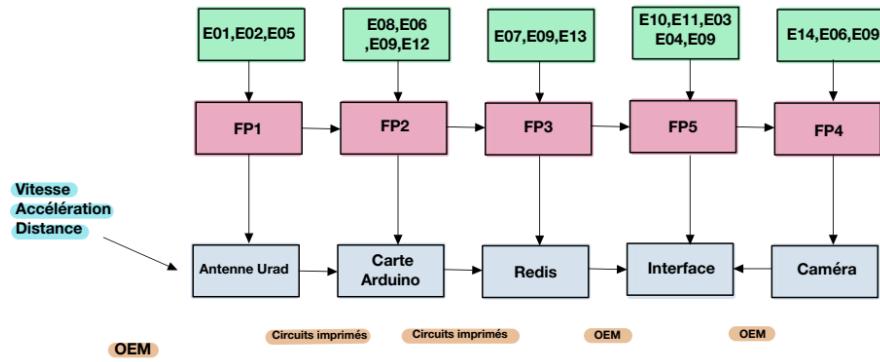


FIGURE 7 – Architecture physique du S4

Cette dernière illustration correspond à notre architecture physique préliminaire. Elle nous permet d'analyser les spécifications des fonctions principales et de comprendre leur interaction avec les sous-systèmes physiques.

De surcroît, nous l'avons mise à jour en accord avec notre nouveau cahier des charges. Nos fonctions principales sont aussi bien mis en valeur dont l'objectif primaire pour notre projet est le renvoi de la distance et la vitesse du coureur avec la meilleure précision possible.

4 Développement

4.1 Introduction de la partie technique

Nous avons voulu ce semestre mettre en réseau nos trois radars uRAD pour essayer de développer un système mesurant les vitesses et les positions de coureurs à pied (pour les fins de courses et donc les photos-finish notamment). Les principales contraintes étaient :

- La simplicité d'utilisation : pour qu'un professeur de sport ou un coach sportif sans connaissances particulières dans le domaine du radar puisse utiliser notre système.
- Le faible coût du système dans son ensemble : pour être en accord avec nos radars qui sont de bonne qualité mais abordables.
- La résolution des mesures : avoir une résolution inférieure à 0.5m/s pour avoir des mesures adaptées à la course à pied.
- Le temps réel : en imaginant que l'utilisateur du système regarderait les mesures en même temps que la course se déroulerait
- L'ajout d'une vidéo : pour un meilleur rendu visuel pour l'utilisateur

Nous avons cherché d'abord à n'utiliser que des radars, nous devions alors localiser les coureurs dans un plan (O, x, y), et tracer les points (x, y) correspondant à des coureurs sur un graphe, comme une vue de haut de la piste.

Cette méthode n'a pas fonctionné comme on le verra par la suite, c'est pourquoi nous nous avons ajouté une caméra au système pour utiliser de la détection vidéo et l'associer aux détections des radars.

Avant toute chose, il faut bien préciser que nos radars ne renvoient pas d'angle, mais seulement des points distance/vitesse. Comme expliqué dans le premier rapport, il est possible d'obtenir des points distance/vitesse directement avec nos radars, car un peu de traitement du signal est intégré sur les cartes électroniques. Dans cette première partie du projet (la trilateration), nous avons utilisé ces points renvoyés par les radars, puis dans un second temps nous avons remis notre traitement du signal pour calculer avec une meilleure résolution les vitesses (nous aborderons le traitement du signal dans la partie suivante).

4.1.1 Trilateration

Nous avons donc d'abord essayé de localiser des coureurs sur un plan (O, x, y) (le plan de la piste) avec uniquement nos radars, mais avec une vidéo filmant la piste. L'utilisateur trace les axes du plan (O, x, y) sur la vidéo, et les points (x, y) qui correspondent aux mesures du radar sont placés dans ce plan sur la vidéo ou sur un graphe.

Nos radars nous donnent des distances pour chaque cible qu'ils détectent, il est donc possible de tracer un cercle de rayon $r = d_{mesuree}$ et de centre $O = (x_{radar}, y_{radar})$.

Si un autre radar détecte la même cible, alors un autre cercle peut être tracé et l'intersection des deux cercles donne un point (x, y) correspondant à une cible. On peut associer à ce point une vitesse, car chaque mesure d'un radar (donc chaque cercle) est une mesure distance/vitesse : une estimation de sa vitesse est donc : $v = \frac{vitesse_{cercle_1} + vitesse_{cercle_2}}{2}$.

On peut ajouter un troisième radar, pour avoir de la trilateration à la manière des GPS : on a alors trois intersections (mesure radar 1 - mesure radar 2, mesure radar 2 - mesure radar 3, mesure radar 1 - mesure radar 3). Ces trois intersections forment un

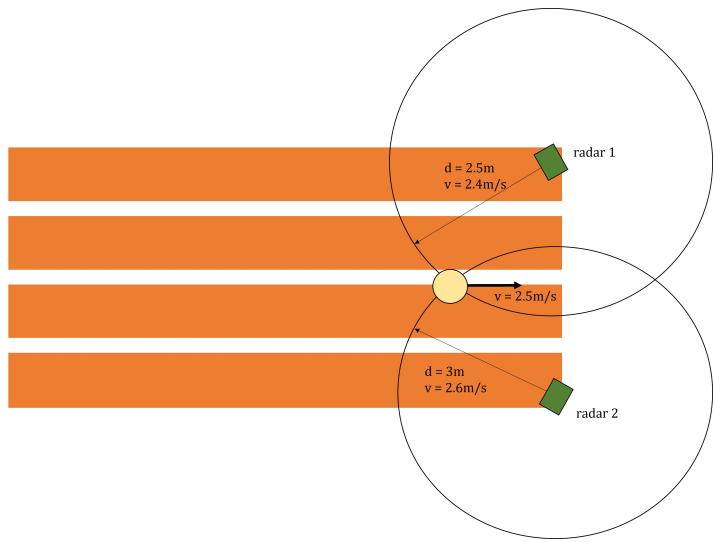


FIGURE 8 – Localisation des coureurs dans un plan (O, x, y) avec deux radars

triangle qui donne une incertitude spatiale d'un point (x, y) . La vitesse du point est encore la moyenne des trois vitesses mesurées.

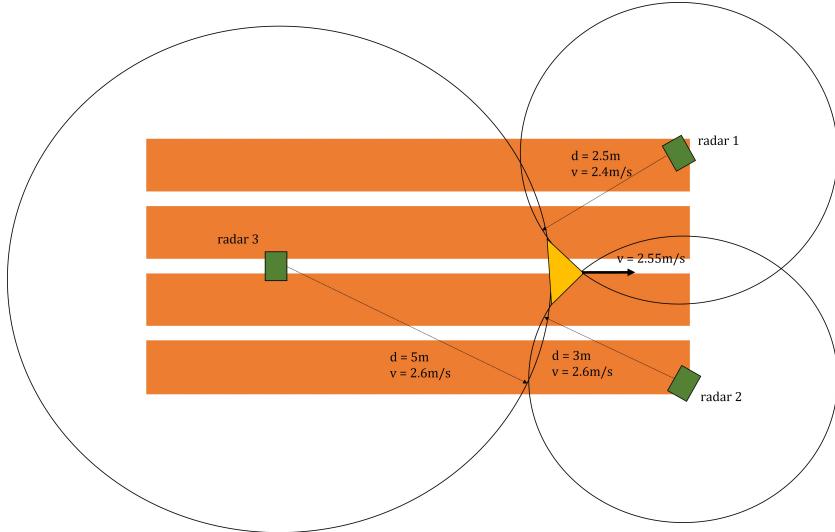


FIGURE 9 – Trilateration

Lorsque plusieurs cibles sont sur la piste, plusieurs cercles sont tracés par radar et il faut former des triangles à partir des cercles du radar 1, du radar 2 et du radar 3. Plusieurs combinaisons de triangles sont possibles et on peut prendre par exemple celle avec l'aire moyenne des triangles la plus faible. Encore mieux que l'aire, on peut choisir la combinaison avec l'écart-type moyen le plus faible (c'est-à-dire avec l'incertitude spatiale moyenne la plus faible).

Une fois la combinaison de triangles obtenue, on a un ensemble des points (x, y) contenant chacun une information de vitesse.

Nous avons essayé la trilateration dans le gymnase de l'ENSTA Bretagne. Après plusieurs essais, nous nous sommes rendus compte que nous avions du mal à avoir des mesures simultanées de la part des trois radars. Cela est dû à l'angle d'ouverture des radars, qui est de 15 degrés : à 20 mètres, la largeur du faisceau est de 5 mètres, ce qui veut dire que les trois radars ne peuvent couvrir qu'une surface circulaire d'un diamètre égal à 5 mètres.

Un autre problème secondaire s'est posé : l'utilisateur doit pour la trilateration tracer un repère du plan (O, x, y) pour indiquer à la vidéo où placer les points (x, y) . Seulement, à cause de la distorsion naturelle due à l'effet de perspective, il est très compliqué de placer correctement les points sur la vidéo.

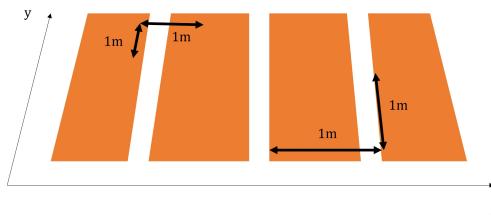


FIGURE 10 – Illustration de la distorsion

Nous cherchons nous à couvrir une piste sur au moins 20 mètres, donc une surface rectangulaire de 10x20 mètres au moins : nous nous en sommes venus au fait que la trilateration n'était pas une bonne manière de procéder pour notre projet, car les radars sont très directifs et la distorsion nous empêche de visualiser facilement les points sur une vidéo.

4.1.2 Superposition

Comme nous l'avons vu, essayer de couvrir une même zone avec plusieurs radars n'est pas adapté pour notre projet.

Nos radars étant très directifs, et ayant une très bonne portée (supérieure à 100 mètres, voir rapport 1), nous les avons donc mis tous dans l'axe de la piste de l'ENSTA Bretagne, chacun couvrant deux couloirs.

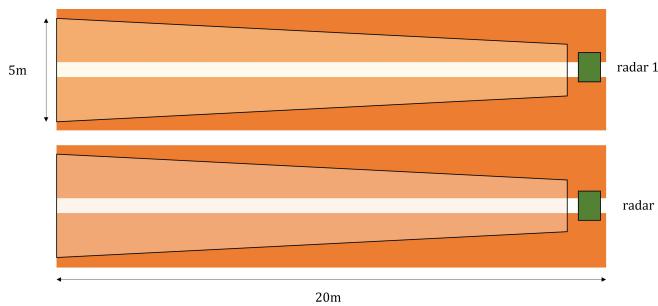


FIGURE 11 – Notre disposition finale

Cela fait, nos radars peuvent mesurer indépendamment les coureurs. Pour visualiser les mesures sur la vidéo, l'utilisateur doit indiquer sur la vidéo où se trouve les radars et

les axes vers lesquels ils pointent avec un vecteur unitaire d'un mètre pour chacun. Une mesure distance/vitesse d'un radar est alors transformée en un point en pixels grâce à son vecteur unitaire et à la distance mesurée par le radar.

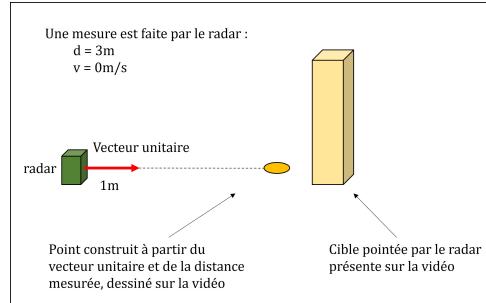


FIGURE 12 – Visualisation des mesures sur la vidéo

Pour améliorer le rendu visuel, nous avons ajouté une détection vidéo (sur laquelle nous reviendrons plus tard) : le point en pixels tracé à la vidéo peut être associé à la détection vidéo, et ainsi afficher les informations de vitesse et de distance par rapport à la ligne d'arrivée d'un coureur directement sur le rectangle de détection vidéo qui englobe le coureur.

Nous nous sommes ainsi rabattus sur cette solution plus simple pour avoir un système qui remplisse le cahier des charges.

4.2 Réseau

4.2.1 Présentation du réseau

Le réseau local est créé à partir d'un routeur WiFi D-Link. Le réseau est constitué d'un serveur Redis, et de clients :

- des radars uRAD portés sur Raspberry Pi 4
- un ordinateur pour l'interface graphique
- un smartphone pour la vidéo

Un serveur Redis est une base de données non persistantes qui stocke des données selon un système clé - valeur. Un client dépose une valeur avec une certaine clé, l'autre client peut récupérer la valeur en indiquant au serveur la clé. Le serveur est hébergé sur une des Raspberry Pi. Redis est connu pour ses très hautes performances, et nous permet de minimiser les temps d'aller-retour serveur-client.

Les requêtes au serveur peuvent s'effectuer depuis un script Python grâce à une librairie Python, et sont très simples à envoyer :

- On instancie un objet Redis chez tous les clients avec l'adresse IPv4 de la Raspberry Pi qui héberge le serveur

```
redis = Redis(host=ip, port=6379)
```
- un radar dépose les données brutes sur le serveur à la clé "donnees"

```
redis.set("donnees", donnees)
```
- le client récupère les données connaissant la clé

```
donnees = redis.get("donnees")
```



FIGURE 13 – Redis

4.2.2 Fonctionnement du réseau

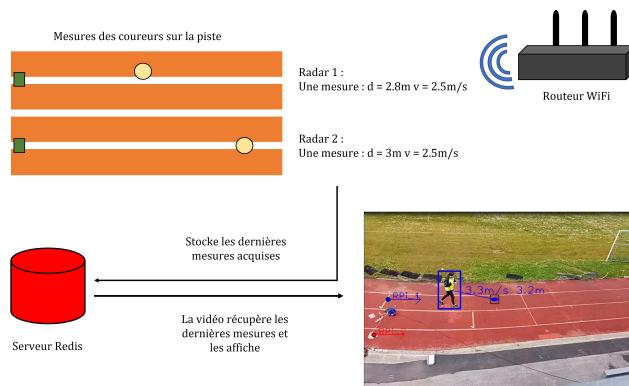


FIGURE 14 – Le réseau local

Le routeur WiFi crée un réseau local, et les radars sont branchés directement en ethernet sur le routeur. Le smartphone, qui se situe en hauteur pour filmer correctement la scène, se connecte au réseau grâce au WiFi émis par le routeur. Enfin, l'ordinateur peut se brancher en ethernet ou bien se connecter avec le WiFi.

Un des radars héberge le serveur Redis. A chaque fois qu'un radar fait une acquisition (toutes les 120ms en moyenne), la Raspberry Pi qui porte le radar fait le traitement du signal pour obtenir les points distance/vitesse à partir des données brutes I et Q (voir le premier rapport). Ces points distance/vitesse sont envoyés sur le serveur.

Le smartphone filme la scène et envoie le flux vidéo sur une page web.

L'utilisateur démarre l'interface graphique sur son ordinateur qui récupère le flux vidéo. Dès lors qu'il a indiqué les vecteurs unitaires de chacun des radars sur la vidéo, la vidéo récupère les points distance/vitesse stockés dans le serveur Redis et les convertit en points en pixels pour les afficher. Enfin ces points en pixels sont associés à des détections vidéo de coureurs.

4.2.3 Latence et fréquence d'acquisition

Comme on l'a dit, le traitement du signal se fait sur les Raspberry Pi et les nouveaux points distance/vitesse sont obtenus toutes les 120ms. Pour un coureur allant à 5m/s (allure de sprint), on réalise en théorie une mesure tous les 60cm, ce qui est adapté pour ce que l'on veut faire.

Sur la vidéo, on peut essayer d'aller chercher à chaque nouvelle frame les points calculés par les Raspberry Pi pour avoir la meilleure fréquence d'acquisition visuelle. Seulement, cet aller-retour avec le serveur peut créer de la latence et faire perdre de la fluidité à la vidéo. Il faut donc trouver expérimentalement un compromis entre nouvelles mesures qui apparaissent sur la vidéo et latence pour avoir le meilleur rendu vidéo possible. Dans les faits, la vidéo va chercher les nouveaux points pour les afficher toutes les cinq frames.

4.3 Traitement du signal et cibles-fantômes

4.3.1 Bref retour sur le semestre 4

Nos radars envoient des ondes continues qui sont modulées en fréquence selon des triangles. La fréquence la plus basse est $f_0 = 24.005GHz$ et la fréquence la plus haute est $f_0 + BW = 24.245GHz$ avec BW la largeur de bande. Les radars retournent un signal complexe d'une durée de 3.5 ms modulé en fréquence par deux triangles (le premier triangle a une durée de 2ms et le second une durée de 1.5ms). Le signal complexe est donc constitué de quatre parties : la rampe montante 1, la rampe descendante 1, la rampe montante 2 et la rampe descendante 2. On peut tracer les quatre spectres de chacune de ces parties : une cible réfléchie par le signal radar donne une fréquence dans chacun des spectres.

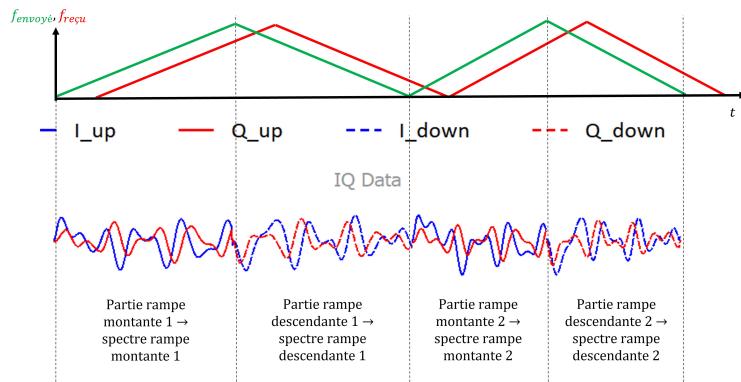


FIGURE 15 – La modulation fréquentielle de nos radars

4.3.2 Associations des fréquences

Lorsque plusieurs cibles sont détectées, les quatre spectres ont eux aussi plusieurs pics. Pour obtenir le point distance/vitesse d'une cible, on doit trouver dans les quatre spectres les quatre fréquences qui ont été produites par la réflexion sur la cible.

A partir d'une seule fréquence obtenue pendant la première rampe montante, on peut tracer une droite $R = f(v)$ sur un graphe distance/vitesse qui nous donne toutes les possibilités de points distance-vitesse pour une seule fréquence. [7][4]

$$R = -\frac{f_0 T_{ramp}}{BW} * v + \frac{c T_{ramp}}{2BW} * f_{mesuree}$$

avec R la distance mesurée par le radar, T_{ramp} la durée d'une rampe qui vaut 1ms si la rampe appartient au premier triangle, et $f_{mesuree}$ la fréquence obtenue dans le spectre. Les autres constantes sont explicitées plus haut.

Une fréquence de la première rampe descendante nous donne une deuxième droite, et l'intersection des deux droites est un point distance/vitesse.

$$R = +\frac{f_0 T_{ramp}}{BW} * v + \frac{c T_{ramp}}{2BW} * f_{mesuree}$$

Cependant, si une autre détection est faite, une nouvelle fréquence apparaît dans chacun des deux spectres. Au total, on peut tracer quatre droites qui forment quatre intersections alors que seulement deux détections ont été faites. Ainsi, pour N cibles réellement détectées, on obtient N^2 cibles, donc $N^2 - N$ cibles fantômes.

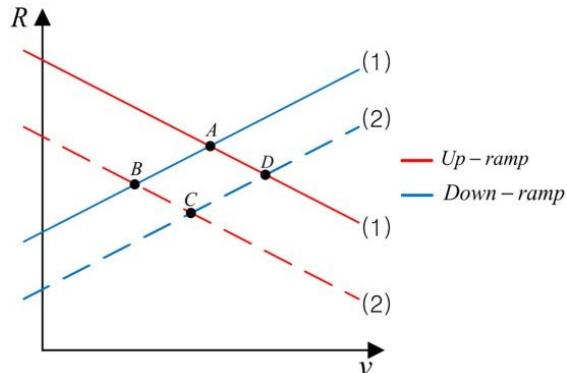


FIGURE 16 – Apparition de cibles-fantômes

Sur la figure 7, on peut voir que deux droites pleines et deux droites en pointillés ont été tracées. Ces couples de droites correspondent chacun à une cible réelle détectée par le radar. Les cibles-fantômes B et D apparaissent lorsque deux droites d'un type différent s'intersectent : elles n'ont pas de réalité physique.

D'où l'intérêt d'ajouter un deuxième triangle de modulation fréquentielle d'une durée plus courte afin que les droites ne sont pas confondues entre elles. Le deuxième triangle permet de confirmer un point d'intersection : si quatre droites (ou seulement trois) s'intersectent en un même point distance/vitesse, alors ce point correspond à une cible réelle.

On voit désormais sur la figure 8 que seuls A et C sont l'intersection de trois droites, donc B et D peuvent être repérés comme étant des cibles-fantômes.

Nous avons donc utilisé cette méthode pour obtenir nos points distance/vitesse, après avoir réalisé une transformée de Fourier rapide sur le signal brut renvoyé par le radar, puis

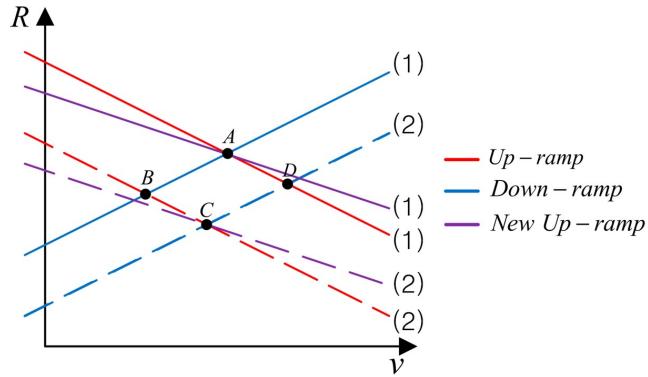


FIGURE 17 – Utilisation d'un deuxième triangle pour supprimer les cibles-fantômes

avoir appliqué l'algorithme CA-CFAR pour trouver les pics du spectre (voir le premier rapport partie traitement du signal).

4.3.3 Zero-padding et résolution en vitesse

Nos radars peuvent renvoyer directement des points distance/vitesse, car le traitement du signal est déjà fait sur leur carte électronique. La fréquence d'acquisition des mesures est optimale, mais la résolution en vitesse moindre : de l'ordre de 0.6m/s. Cette résolution est trop mauvaise pour notre projet, car tous les coureurs auraient eu la même vitesse mesurée.

Si la résolution des vitesses mesurées directement par les radars n'est pas bonne, c'est parce qu'elle est liée au zero-padding (décrit dans le premier rapport), qui est un ajout d'échantillons nuls au signal brut avant de procéder à la transformée de Fourier. Cet ajout de points doit mener le signal brut à avoir une longueur égale à une puissance de deux (pour optimiser la vitesse de calcul de la transformée de Fourier). Ainsi, pour augmenter un peu la résolution, il faut augmenter énormément le nombre d'échantillons et les cartes électroniques des radars n'ont certainement pas la mémoire pour stocker le signal obtenu après un zero-padding trop important.

Nous pouvons nous utiliser nos Raspberry Pi pour faire le traitement du signal et améliorer la résolution sans rencontrer ce problème de mémoire. La résolution en vitesse est liée au nombre d'échantillons par cette formule :

$$\Delta v = \frac{c * (F_e / N_{FFT})}{2f_0}$$

avec F_e la fréquence d'échantillonnage du signal brut, N_{FFT} le nombre d'échantillons après le padding, f_0 la fréquence d'émission la plus basse du signal, et c la vitesse de la lumière. Cette formule est obtenue grâce à la résolution en fréquence du spectre qui vaut tout simplement $\Delta f = \frac{F_e}{N_{FFT}}$ puisque le spectre a une largeur F_e d'après Shannon.

On peut tracer ainsi la courbe reliant résolution en vitesse et nombre d'échantillons du signal avant transformée de Fourier :

Nous avons opté dans nos tests pour fixer le nombre d'échantillons à 4096, donc d'ajouter $4096 - 750 = 3346$ zéros à la fin du signal brut pour une résolution de 0.31m/s.

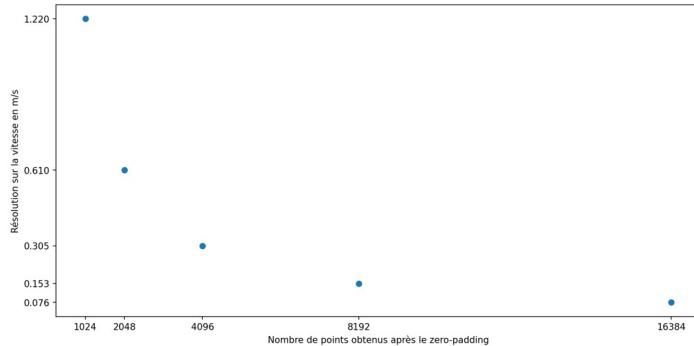


FIGURE 18 – Résolution en vitesse et nombre d’échantillons

Il ne faut pas oublier que l’ajout d’échantillons nuls fait gagner en résolution mais perdre en vitesse de calcul.

4.4 Détection vidéo et suivi de cibles

Nous avons choisi de chercher à valider notre système par un suivi de personnes par vidéo. Cette méthode comprend deux étapes : la détection de personne, pour laquelle nous avons testé plusieurs méthodes, et le suivi de cible. Les détections de personnes sont ensuite associer aux mesures des radars grâce à l’algorithme Hongrois dont nous avons détaillé le fonctionnement dans le premier rapport.

Nous étudierons tout d’abord les différentes méthodes de détection d’objets testées, à savoir les histogrammes de gradient orienté, un réseau neuronal convolutif et la soustraction d’arrière plan, avant de nous intéresser au filtre de Kalman.

4.4.1 La détection d’objets

Nos recherches nous ont conduit à nous intéresser, pour la détection, aux histogrammes de gradient orientés conformément à [4] et [2], ainsi qu’aux réseaux de neurone convolutif You Only Look Once, en accord avec [6] et [1]. Enfin, nous nous sommes tournés vers la soustraction d’arrière-plan.

Les histogrammes de gradient orienté La détection d’objet par histogramme de grandient orienté (HOG) passe par deux étapes : Tout d’abord, il s’agit de créer histogramme de l’orientation du gradient des images considérées, puis d’assurer la classification supervisée des objets caractérisés au préalable, à l’aide de machines à vecteurs de support. On utilisera ici le modèle préalablement implémenté sur la librairie OpenCV.

Les réseaux neuronaux convolutifs On choisit d’utiliser le système d’implémentation de réseau neuronal convolutif YOLO pour You Only Look Once, décrit dans [3].

Les réseaux de neurones convolutifs ont la particularité de se décomposer en deux blocs de réseau de neurones, le premier, spécifique aux réseaux de neurones convolutifs, ayant pour rôle d’extraire les caractéristiques des données d’entrées et fournissant ses

conclusions au deuxième bloc, lequel utilisera les dites caractéristiques pour classifier les données d'entrées.

L'implémentation de réseaux YOLO consiste à prédéfinir des cadres de délimitations qui correspondront à chaque objet d'intérêt, et des identifier sur l'image traitée des anchor box, susceptibles de contenir des objets d'intérêt, avant de classifier ces dernières dans la deuxième bloc du système, présenté précédemment.

La soustraction d'arrière-plan La méthode de soustraction d'arrière plan consistera à créer un masque composé des pixels invariants d'une image à l'autre de la vidéo, ce qui permet de définir les contours des objets mobiles de notre vidéo. On utilise là encore les fonctions implémentées dans la librairie OpenCV.

Pour départager nos modèles, nous avons pris en compte leurs performances, mais également la rapidité des calculs. En effet, YOLO est très efficace pour différencier des personnes proches l'une de l'autre, cependant le nombre de faux positifs est très important. Si l'utilisation de HOG ne permet pas aussi bien de distinguer des cibles proches, il possède néanmoins un taux de faux-positifs bien inférieur dans nos essais. Le nombre de frame per second (fps) moyen pour chacune de ces deux méthodes était respectivement de 22 et 25. Par conséquent, nous avons choisi de d'utiliser la méthode de soustraction d'arrière-plan qui, si elle possède des capacités de distinction inférieures aux deux autres méthodes, permet d'attendre 56 fps en moyenne.

4.4.2 Le suivi de cibles par filtre de Kalman

Nous avons choisi d'implémenter le suivi de cible par un filtre de Kalman, en suivant l'exemple de [5]. Nous n'avons cependant pas pu pour l'instant expérimenter les résultats de son implémentation.

4.5 Interface graphique

4.5.1 Technologies utilisées

L'interface graphique a été réalisée avec des technologies Web : HTML/CSS/Javascript. Ce choix a été fait par rapport à Qt ou JavaFX pour plusieurs raisons :

- On pensait à l'origine ne pas se servir de Redis et faire un vrai serveur Web, c'est-à-dire envoyer les données des radars avec des requêtes HTTP, ce qui aurait été beaucoup plus long que d'utiliser Redis
- La possibilité créatrice est plus grande avec le Web
Par exemple, sur notre interface graphique, l'utilisateur doit placer des vecteurs sur la vidéo avec la souris. Avec Qt, cette interaction est peut-être implémentable mais plus difficilement.
- L'utilisateur n'a rien à télécharger
- L'interface graphique est disponible sur toutes les plateformes (tablette, smartphone, et pc quelque soit le système d'exploitation)

Le principal défaut est la lenteur par rapport à une simple interface graphique faite avec OpenCV, parce qu'on doit envoyer le flux vidéo depuis OpenCV au serveur Web dans notre cas, alors qu'on pourrait directement l'afficher avec OpenCV.

En plus de l'affichage de la vidéo, les transformées de Fourier rapides et la méthode d'associations des fréquences peuvent être tracées dans des graphes en parallèle. Nous avons utilisé la librairie Plotly pour tracer ces graphes.

4.5.2 Présentation de l'interface graphique

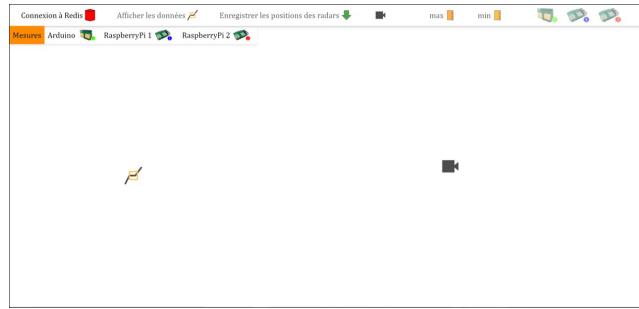


FIGURE 19 – La page principale

L'utilisateur doit d'abord se connecter au serveur Redis : c'est tout simplement un ping qui est fait au serveur Redis, et si le serveur Redis répond alors l'utilisateur peut poursuivre sur le site, sinon il peut retenter de se connecter au serveur.

Ensuite, l'utilisateur peut lancer la vidéo en cliquant sur le logo caméra, puis placer les icônes des radars sur la vidéo aux emplacements où ils se trouvent dans la réalité. Pour chaque radar placé, il doit cliquer sur la vidéo pour indiquer le deuxième point du vecteur. Une fois le vecteur tracé, l'utilisateur indique la longueur réelle du vecteur tracé en mètres.

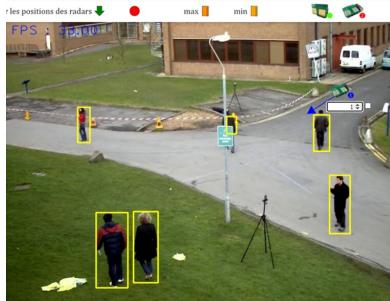


FIGURE 20 – L'utilisateur indique à la vidéo la longueur du vecteur qu'il vient de tracer

Dès lors que les vecteurs ont été tracés et leurs longueurs indiquées, l'utilisateur peut enregistrer les positions des radars : les mesures des radars sont affichées et associées aux détections vidéo.

S'il le souhaite, l'utilisateur peut observer les transformées de Fourier qui sont calculées par les radars en activant la visualisation des données. Il peut aussi voir la méthode d'association des fréquences par intersection de droites dont nous avons parlé plus tôt.

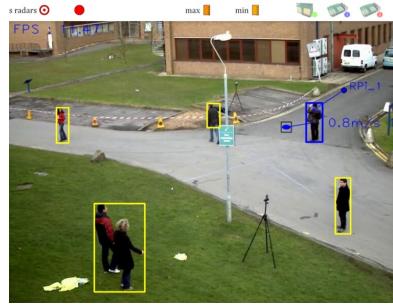


FIGURE 21 – Les mesures sont affichées dès que l'utilisateur a enregistré les positions des radars

4.6 Résultats et pistes d'amélioration

4.6.1 Présentation des résultats

Nous avons pu prendre des mesures sur des coureurs qui faisaient leur footing entre midi et deux. L'un des coureurs était plus rapide que l'autre donc ses mesures sont plus espacées spatialement.

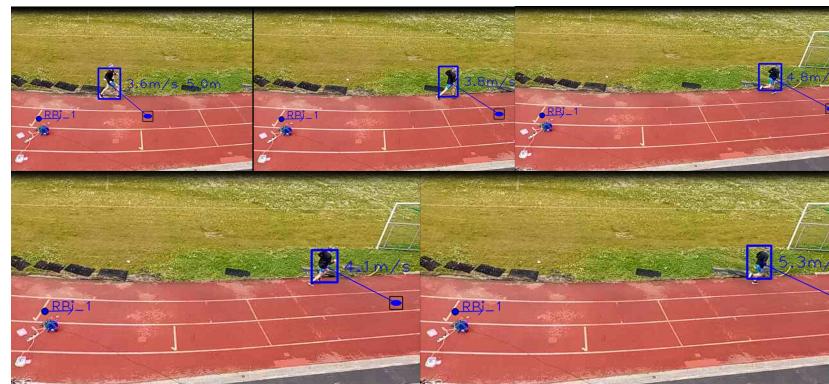


FIGURE 22 – Coureur rapide



FIGURE 23 – Coureur lent

Comme on peut le voir, le résultat est plutôt bon car sur une dizaine de mètres, nous avons pu prendre 5 mesures pour le coureur rapide et une dizaine de mesures pour le coureur lent, et les mesures de vitesse ont une bonne résolution (jusqu'à 0,2m/s). Les radars ayant une portée bien plus grande que la dizaine de mètres de piste filmée, nous sommes uniquement limités par l'angle d'ouverture de la caméra du téléphone portable.

4.6.2 Pistes d'amélioration

Améliorations simples

Nous proposons deux pistes d'amélioration simples possibles pour perfectionner le projet. La première est l'utilisation d'une caméra de type caméra de vidéo-surveillance disposant d'un très grand angle d'ouverture. On pourrait ainsi voir la piste sur toute sa partie droite sur la vidéo, et donc utiliser pleinement la portée des radars. Petit bémol, la vidéo serait distordue aux extrémités, mais on devrait pouvoir en théorie corriger cette distorsion grâce à un algorithme. Autre point négatif, une caméra demi-sphérique serait compliquée à positionner sur le toit d'un bâtiment comme on le fait avec nos smartphones et un trépied, il faudrait passer du temps à concevoir un système permettant de fixer la caméra correctement.

La seconde voie d'amélioration est l'achat d'un routeur WiFi haut de gamme (le nôtre est le moins cher de sa gamme), qui permettrait d'augmenter les débits et donc de fluidifier la vidéo quand bien même la récupération des mesures se ferait à chaque frame de la vidéo.

Fréquence d'apparation des points optimale et flux vidéo sans latence

Dans le cadre d'un projet plus scientifique, où l'on ne chercherait pas à produire un logiciel simple d'utilisation, on pourrait seulement faire une interface avec OpenCV en C++, et coder l'association détections vidéo - mesures des radars en C++, pour optimiser la vitesse d'exécution.

Côté radar et RaspberryPi, on pourrait faire tout le traitement du signal et l'association des fréquences en C/C++ avec le package Weave fournit par Scipy.

On peut imaginer qu'avec de telles améliorations qu'on aurait une fréquence d'apparition des points sur la vidéo optimale et un flux vidéo sans trop de latence, cependant cela ne rentrerait pas dans notre cahier des charges qui est d'avoir un logiciel adapté à un utilisateur sans connaissances dans l'informatique.

5 Tests réalisés

5.1 Tests unitaires

Nous avons réalisé de nombreux tests unitaires, surtout sur la partie traitement du signal afin de vérifier que nos fonctions retournent finalement les bonnes mesures de distance et de vitesse. Nous détaillons ici les tests unitaires que nous avons réalisé sur nos algorithmes à seuils adaptatifs (CA-CFAR, voir le premier rapport) pour la détection de pics dans un spectre.

5.1.1 Détail du test unitaire pour notre algorithme à seuil adaptatif

Nous avons pour tester notre algorithme CA-CFAR utiliser une fonction de MATLAB qui implémente cet algorithme, et nous avons comparé les seuils de notre algorithme à ceux de la fonction de MATLAB.

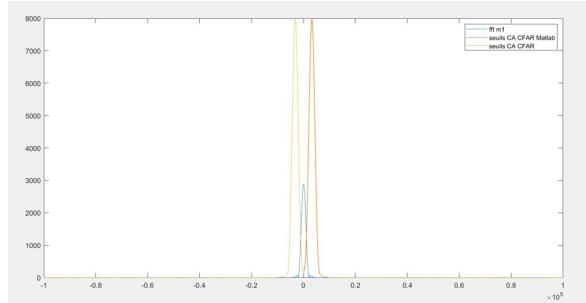


FIGURE 24 – Test unitaire de notre algorithme à seuil adaptatif

On a bien observé bien que les seuils sont confondus, donc notre algorithme codé en Python pu être validé après ce test. Le code du test est disponible en annexe et est exécutable avec MATLAB.

5.1.2 Détail du test unitaire pour notre algorithme d’associations de fréquences

Pour s’assurer que notre algorithme fonctionnait bien, nous avons testé qu’on trouvait bien pour N points distance/vitesse N^2 intersections entre les droites venant de la première modulation fréquentielle et N^2 autres intersections entre les droites venant de la deuxième modulation fréquentielle.

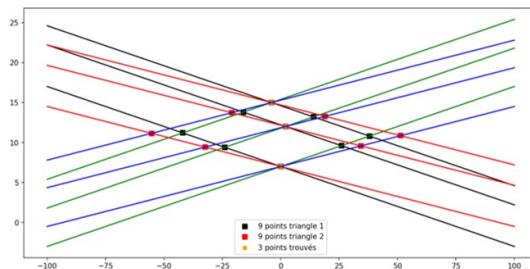


FIGURE 25 – Test unitaire de notre algorithme à seuil adaptatif

Le code du test est disponible en annexe et est exécutable avec Python.

5.2 Tests d’intégration

Nous n’avions eu que peu d’intégration de code à faire étant donné que nous avons fonctionné par blocs : un bloc principal d’interface graphique et de traitement du signal, un bloc de détection vidéo et un bloc de suivi de cibles.

5.2.1 Test d'intégration de la détection vidéo

La détection vidéo doit être une fonction qui prend en argument une frame de la vidéo et qui renvoie des rectangles correspondant à des détections de personnes sur cette frame.

Nous nous sommes préalablement mis d'accord sur la forme des rectangles que le détecteur devait renvoyer : Des dictionnaires Python de la forme :

```
rectangle = "x" : x, "y" : y, "h" : h, "w" : w
```

Avec (x, y) la position du pixel tout en haut à gauche du rectangle, et h et w la hauteur et la largeur du rectangle en pixels.

5.2.2 Test d'intégration du suivi de cibles

Nous avons fonctionné de la même manière pour le suivi de cibles : le suivi de cibles doit prendre en argument la liste des détections et renvoyer une liste de cibles que l'on doit afficher.

Les cibles sont des instances d'une classe Python que nous avons défini ensemble, qui contient comme variable d'instance une position (x, y) qui varie et un identifiant qui reste constant.

5.3 Test de validation fonctionnelle

Exigences	Validation
Résolution des vitesses < 0.5m/s	Oui
Vidéo fluide	20 FPS (fluidité parfaite à 30 FPS)
Couvre toute la piste	Oui
Simple d'utilisation	Oui avec explications (testé avec M. Tanguy)
Abordable	Coût inférieur à 1000 euros

FIGURE 26 – Notre test de validation fonctionnelle

Conclusion

Nous avons ainsi réussi à créer un système fonctionnel permettant de mesurer les vitesses de coureurs avec une bonne résolution. Notre système est abordable, simple à installer et simple d'utilisation, afin qu'un professeur de sport ou qu'un coach sportif puisse s'en servir pour les entraînements qu'il organise ou dans le cadre d'une compétition. Nous avons utilisé des technologies diverses pour appliquer des connaissances vues en cours et pour apprendre un maximum de choses durant ce projet, que ce soit au niveau des langages de programmation (Python, MATLAB, Javascript), du réseau (Redis, routeur Wi-Fi), de la détection vidéo (OpenCV et YOLO) ou encore du suivi de cibles (algorithme Hongrois, filtre de Kalman). Nous espérons que le travail rendu donnera la juste mesure du travail accompli avec passion durant ce projet.

A Annexe

A.1 Code du test unitaire de l'algorithme CA-CFAR (MATLAB)

```
f0 =24.005e9;
BW = 240e6;
n = 2048;
Ns = 200;
c = 3e8;
Fs = 200000;
max_voltage = 3.3;
ADC_bits = 12;
ADC_intervals = 2^ADC_bits;

I = [2045 2048 2046 2043 2045 2040 2042 2040 2036 2043 2038 2040 2037 2034
2040 2036 2038 2042 2039 2043 2035 2038 2042 2035 2039 2040 2040 2045
2035 2041 2042 2038 2043 2041 2046 2049 2042 2046 2048 2050 2053 2048
2051 2049 2048 2052 2052 2058 2062 2058 2061 2059 2060 2059 2053 2058
2060 2056 2057 2055 2058 2055 2052 2057 2053 2052 2049 2047 2053 2045
2044 2045 2045 2049 2045 2044 2049 2045 2045 2044 2046 2050 2044 2043
2045 2044 2045 2040 2043 2047 2044 2043 2042 2044 2046 2042 2046 2048
2046 2047 2046 2052 2052 2048 2051 2050 2051 2049 2045 2050 2053 2050
2050 2049 2054 2052 2048 2051 2054 2053 2052 2049 2053 2051 2048 2049
2048 2049 2048 2047 2048 2046 2044 2041 2041 2043 2037 2036 2041 2042
2041 2039 2038 2043 2039 2042 2046 2044 2046 2043 2046 2050 2046 2046
2047 2047 2048 2044 2045 2047 2044 2044 2043 2047 2048 2044 2046 2048
2049 2048 2045 2048 2047 2045 2047 2046 2047 2048 2045 2048 2049 2049
2049 2050 2056 2059 2057 2059 2060 2061 2056 2051 2052 2052 2049 2050
2049 2052 2051 2028 2045 2047 2050 2050 2042 2046 2049 2046 2047 2047
2050 2052 2046 2049 2052 2052 2053 2052 2060 2061 2054 2055 2057 2060
2054 2053 2057 2056 2054 2053 2052 2055 2050 2046 2047 2045 2045 2043
2045 2048 2041 2039 2040 2041 2044 2038 2039 2042 2037 2039 2040 2042
2045 2045 2046 2041 2039 2038 2038 2043 2044 2046 2048 2049 2054 2051
2050 2052 2051 2053 2053 2053 2057 2055 2053 2054 2054 2055 2053 2053
2057 2055 2056 2054 2055 2055 2052 2053 2051 2048 2046 2042 2046 2041
2042 2045 2045 2047 2044 2043 2044 2039 2042 2046 2046 2050 2047 2047
2049 2048 2050 2048 2049 2054 2049 2051 2050 2050 2050 2043 2046 2047
2044 2046 2042 2046 2046 2041 2047 2044 2046 2046 2042 2049 2047 2047
2052 2050 2053 2049 2046 2051 2049 2050 2051 2049 2053 2048 2047 2051
2049 2052 2047 2049 2054 2049 2050 2048 2047 2050 2042 2049 2049 2044
2045 2041 2047 2047 2040 2048 2045 2045 2047 2042 2047 2044 2043 2049
2041 2045 2044 2043 2050 2042 2043 2045 2043 2050 2042 2045 2050 2044
2049 2047 2047 2050 2044 2045 2046 2050 2050 2044 2049 2044 2046 2045
2043 2050 2044 2044 2046 2045 2049 2044 2050 2056 2047 2050 2048 2050
2052 2045 2050 2054 2053 2057 2050 2057 2057 2050 2055 2056 2058 2058
2051 2054 2053 2049 2048 2043 2046 2046 2044 2049 2046 2046 2046 2042
```

```

2047 2043 2040 2042 2040 2044 2041 2042 2047 2043 2044 2046 2047 2051
2047 2047 2053 2053 2054 2051 2049 2053 2047 2047 2049 2052 2052 2057 2052
2052 2053 2047 2048 2047 2051 2053 2048 2047 2050 2052 2051 2047 2051
2054 2049 2050 2051 2051 2050 2045 2049 2052 2046 2044 2045 2049 2048
2043 2047 2047 2043 2046 2045 2051 2051 2045 2047 2047 2047 2048 2044 2040
2045 2044 2043 2041 2043 2047 2043 2044 2051 2050 2050 2046 2048 2055
2052 2050 2054 2053 2053 2048 2049 2055 2052 2050 2050 2054 2059 2050
2049 2055 2055 2055 2057 2053 2056 2057 2054 2056 2055 2054 2056
2050 2050 2049 2047 2049 2045 2048 2048 2044 2045 2042 2044 2043 2040
2044 2043 2044 2045 2044 2046 2044 2043 2047 2044 2046 2044 2042 2047
2043 2044 2045 2045 2048 2043 2044 2044 2044 2046 2044 2049 2052 2051
2052 2052 2054 2055 2052 2056 2056 2055 2056 2053 2055 2054 2052 2053
2049 2053 2051 2050 2049 2047 2051 2050 2051 2053 2049 2052 2054 2052
2055 2051 2052 2052 2048 2050 2048 2046 2046 2043 2048 2047 2045 2047
2042 2044 2042 2042 2046 2040 2042 2042 2039 2044 2043 2046 2048 2044
2047 2045 2047 2049 2046 2050 2047 2047 2049 2046 2052 2053 2049 2052
2046 2046 2048 2040 2048 2047 2051 2054 2045 2045 2039 2039 2048 2042
2048 2046 2043 2053 2049 2050 2050 2046 2053 2047 2047 2049 2046 2045 2045];

```

```

Q = [2050 2052 2053 2053 2052 2052 2054 2052 2051 2049 2048 2048 2044 2042
2044 2042 2039 2038 2042 2045 2042 2046 2049 2050 2050 2049 2053 2053
2048 2049 2050 2053 2050 2047 2051 2048 2046 2045 2047 2053 2051 2052
2054 2053 2053 2052 2054 2059 2059 2057 2055 2049 2045 2039 2042 2045
2044 2045 2045 2046 2044 2041 2044 2044 2044 2046 2046 2048 2048 2046
2051 2051 2051 2048 2046 2053 2051 2048 2049 2049 2050 2046 2043 2047
2046 2044 2041 2040 2041 2036 2036 2041 2041 2043 2040 2041 2046 2044
2044 2045 2048 2050 2047 2050 2055 2053 2054 2055 2059 2061 2054 2058
2058 2060 2057 2053 2062 2060 2055 2055 2054 2058 2055 2051 2054 2052
2051 2049 2049 2051 2047 2046 2049 2045 2042 2036 2037 2043 2040 2038
2039 2042 2043 2038 2036 2042 2043 2044 2041 2043 2047 2043 2042 2042
2045 2046 2039 2042 2048 2046 2047 2046 2051 2054 2050 2053 2054 2056
2059 2058 2063 2062 2060 2062 2058 2058 2054 2054 2058 2051 2048 2050
2050 2050 2045 2042 2046 2045 2043 2038 2039 2044 2040 2034 2036 2037
2035 2034 2040 2038 2050 2052 2054 2054 2051 2051 2052 2047 2044 2044
2047 2047 2045 2045 2048 2045 2043 2037 2042 2046 2043 2045 2049 2050
2048 2046 2049 2049 2048 2048 2050 2052 2051 2046 2047 2047 2046 2044
2041 2044 2041 2039 2042 2047 2049 2043 2042 2045 2044 2042 2041 2043
2045 2045 2043 2043 2045 2044 2045 2047 2045 2045 2048 2055 2060 2058
2057 2055 2055 2053 2053 2054 2052 2049 2050 2051 2047 2045 2047 2047
2049 2044 2043 2049 2047 2044 2041 2043 2049 2046 2046 2048 2048 2048
2046 2049 2051 2049 2048 2050 2051 2051 2045 2050 2051 2048 2045 2045
2047 2045 2045 2047 2046 2044 2043 2045 2051 2051 2046 2047 2046 2047
2044 2045 2051 2048 2043 2044 2048 2051 2047 2045 2050 2050 2050 2048
2050 2053 2051 2052 2055 2054 2054 2052 2058 2060 2056 2056 2057 2058
2058 2055 2057 2055 2055 2057 2053 2053 2052 2046 2046 2045 2047 2047
2046 2050 2050 2050 2049 2047 2049 2048 2047 2047 2046 2047 2047 2045 2042

```

```

2045 2047 2045 2044 2043 2046 2042 2038 2038 2042 2043 2042 2042 2040 2043
2043 2040 2041 2042 2043 2039 2041 2050 2050 2050 2052 2055 2053 2053
2059 2059 2057 2054 2053 2057 2052 2051 2053 2053 2051 2047 2050 2052
2048 2047 2047 2047 2046 2041 2042 2042 2038 2035 2035 2039 2040 2037
2042 2044 2045 2043 2043 2051 2051 2045 2044 2046 2049 2043 2040 2045
2045 2043 2042 2043 2047 2043 2041 2042 2042 2043 2043 2045 2046 2044
2045 2050 2048 2049 2045 2048 2053 2051 2048 2051 2052 2052 2051 2060
2067 2066 2061 2059 2060 2059 2056 2058 2060 2062 2057 2054 2060 2059
2056 2056 2058 2060 2056 2052 2056 2058 2058 2053 2054 2059 2056 2054
2053 2051 2050 2046 2046 2049 2047 2046 2045 2046 2047 2044 2044 2046
2043 2041 2040 2045 2046 2042 2044 2043 2043 2038 2036 2041 2040 2036
2037 2037 2038 2033 2027 2033 2033 2030 2032 2041 2041 2037 2037
2040 2040 2035 2043 2043 2043 2048 2047 2047 2047 2047 2049 2044 2044
2047 2049 2050 2050 2053 2056 2051 2052 2054 2054 2057 2055 2056 2055
2052 2050 2048 2048 2044 2039 2043 2043 2041 2039 2040 2047 2046 2040
2044 2044 2042 2037 2040 2045 2041 2041 2046 2053 2057 2052 2052 2056
2056 2057 2055 2055 2056 2051 2054 2055 2053 2049 2047 2050 2051 2044
2045 2046 2048 2047 2043 2045 2047 2044 2042 2042 2043 2044 2041 2043
2044 2045 2042 2043 2046 2046 2044 2045 2048 2049 2044 2044 2051 2053
2051 2049 2052 2058 2054 2052 2054 2054 2056 2055 2057 2060 2057 2056
2057 2057 2057 2052 2055 2057 2054 2052 2051 2053 2054 2049 2050 2049
2046 2045 2045 2047 2046 2045 2043 2044 2045 2042 2037 2036 2040 2042
2042 2044 2049 2050 2046 2045 2046 2048 2042 2038 2041 2043 2041 2050];

```

```

I_m1 = I(1:Ns);
Q_m1 = Q(1:Ns);
I_d1 = I(Ns+1:2*Ns);
Q_d1 = Q(Ns+1:2*Ns);
I_m2 = I(2*Ns+1:2.75*Ns);
Q_m2 = Q(2*Ns+1:2.75*Ns);
I_d2 = I(2.75*Ns+1:3.5*Ns);
Q_d2 = Q(2.75*Ns+1:3.5*Ns);

vecteur_complexe_m1 = I_m1 + 1j*Q_m1;
vecteur_complexe_d1 = I_d1 - 1j*Q_d1;
vecteur_complexe_m2 = I_m2 + 1j*Q_m2;
vecteur_complexe_d2 = I_d2 - 1j*Q_d2;

hann_1 = hann(Ns).';
hann_2 = hann(0.75*Ns).';

vecteur_complexe_m1 = vecteur_complexe_m1.*hann_1;
vecteur_complexe_d1 = vecteur_complexe_d1.*hann_1;
vecteur_complexe_m2 = vecteur_complexe_m2.*hann_2;
vecteur_complexe_d2 = vecteur_complexe_d2.*hann_2;

```

```

fft_m1 = 2*abs(fftshift(fft(vecteur_complexe_m1/Ns, n)));
fft_d1 = 2*abs(fftshift(fft(vecteur_complexe_d1/Ns, n)));

fft_m2 = 2*abs(fftshift(fft(vecteur_complexe_m2/(0.75*Ns), n)));
fft_d2 = 2*abs(fftshift(fft(vecteur_complexe_d2/(0.75*Ns), n)));

frequences = linspace(-Fs/2, Fs/2, n);
n_train = 50;
cfar = phased.CFARDetector("NumTrainingCells", n_train, "NumGuardCells", 40, ...
    "Method", "CA", ...
    "ThresholdOutputPort", true, "ProbabilityFalseAlarm", 0.001);

[pics_m1, seuils_m1] = cfar(fft_m1.', 1:length(fft_m1));

plot(frequences, fft_m1);
hold on;
plot(frequences, seuils_m1);

res1 = pyrun(["import numpy as np", ...
    "from scipy.signal import find_peaks", ...
    "from scipy.ndimage import convolve", ...
    "fa_rate = 0.001", ...
    "n_train = 50", ...
    "n_guard = 40", ...
    "N_FFT = 2048", ...
    "Fs = 200000", ...
    "frequences=[-Fs/2 + ((Fs*i)/N_FFT) for i in range(N_FFT)]", ...
    "alpha = n_train * (fa_rate ** (-1 / n_train) - 1)", ...
    "ones = np.ones(n_train//2)", ...
    "zeros = np.zeros(n_guard)", ...
    "kernel = np.concatenate([ones, zeros, ones])", ...
    "conv = convolve(fft, kernel, mode = 'constant')", ...
    "mean = conv/n_train", "thresholds = mean*alpha", ...
    "peaks, properties = find_peaks(conv, " + ...
    "height = thresholds, distance = (n_train + n_guard)//2)", ...
    "res = {'indices_pics' : peaks.tolist(), " + ...
    "'frequences_pics' : [frequences[i] for i in peaks], " + ...
    "'valeurs_pics' : properties['peak_heights'].tolist(), " + ...
    "'seuils' : thresholds.tolist()}", ...
    "res", fft = fft_m1);

frequences_pics = double(res1{"frequences_pics"});
valeurs_pics = double(res1{"valeurs_pics"});
seuils = double(res1{"seuils"});
plot(frequences, seuils);

```

```
legend("fft m1", "seuils CA CFAR Matlab", "seuils CA CFAR")
```

A.2 Code du test unitaire de l'algorithme d'associations de fréquences (Python)

```
#f_m = (2*BW/(c*T_ramp))*R + ((2*f0)/c)*V

#f_d = -(2*BW/(c*T_ramp))*R + ((2*f0)/c)*V

#R = (c*T_ramp*f_m/(2*BW)) - ((f0*T_ramp)/(BW))*V

#R = (c*T_ramp*f_d/(2*BW)) + ((f0*T_ramp)/(BW))*V

import matplotlib.pyplot as plt
import math
import random
f0 = 24.005e9
BW = 240e6
c = 3e8
T_ramp_1 = 1e-3
T_ramp_2 = 0.75e-3

def obtenir_frequencies(v, d):
    delta_f_1 = d*2*BW/(c*T_ramp_1)
    delta_f_2 = d*2*BW/(c*T_ramp_2)
    f_doppler = 2*f0*v/c
    frequences = {"f_m1" : delta_f_1 + f_doppler, "f_d1" : delta_f_1 - f_doppler,
                  "f_m2" : delta_f_2 + f_doppler, "f_d2" : delta_f_2 - f_doppler}
    return frequences

def obtener_intersection(d_1, d_2):
    v = (d_1["b"] - d_2["b"])/(d_2["m"] - d_1["m"])
    d = d_1["m"]*v + d_1["b"]
    return [v, d]

def associations(frequences_m1, frequences_d1, frequences_m2, frequences_d2):
    intersections_1, intersections_2 = [], []
    droites_m1, droites_d1, droites_m2, droites_d2 = [], [], [], []
    for f_m1 in frequences_m1:
        droite = {
            "m" : -(f0*T_ramp_1)/BW,
            "b" : (f_m1*c*T_ramp_1)/(2*BW)
        }
        droites_m1.append(droite)

    for f_d1 in frequences_d1 :
```

```

droite = {
    "m" : (f0*T_ramp_1)/BW,
    "b" : (f_d1*c*T_ramp_1)/(2*BW)
}
droites_d1.append(droite)

for f_m2 in frequences_m2 :
    droite = {
        "m" : -(f0*T_ramp_2)/BW,
        "b" : (f_m2*c*T_ramp_2)/(2*BW)
    }
    droites_m2.append(droite)

for f_d2 in frequences_d2 :
    droite = {
        "m" : (f0*T_ramp_2)/BW,
        "b" : (f_d2*c*T_ramp_2)/(2*BW)
    }
    droites_d2.append(droite)

for d_m1 in droites_m1:
    for d_d1 in droites_d1:
        intersections_1.append(obtenir_intersection(d_m1, d_d1))

for d_m2 in droites_m2:
    for d_d2 in droites_d2:
        intersections_2.append(obtenir_intersection(d_m2, d_d2))

points = []
for inter_1 in intersections_1 :
    for inter_2 in intersections_2 :
        if math.sqrt((inter_1[0] - inter_2[0])**2 +
                    (inter_1[1] - inter_2[1])**2)<1e-3 :
            points.append(inter_1)
return {
    "points" : points,
    "intersections_1" : intersections_1,
    "intersections_2" : intersections_2,
    "droites" : {
        "droites_m1" : droites_m1,
        "droites_d1" : droites_d1,
        "droites_m2" : droites_m2,
        "droites_d2" : droites_d2,
    }
}

```

```

frequencies_m1, frequencies_d1, frequencies_m2, frequencies_d2 = [], [], [], []

def ajouter_frequencies(frequencies):
    f_m1, f_d1, f_m2, f_d2 = frequencies["f_m1"], frequencies["f_d1"], \
        frequencies["f_m2"], frequencies["f_d2"]
    frequencies_m1.append(f_m1)
    frequencies_d1.append(f_d1)
    frequencies_m2.append(f_m2)
    frequencies_d2.append(f_d2)

vrais_points = []
for i in range(3):
    v, d = random.randint(-5, 5), random.randint(0, 15)
    vrais_points.append([v, d])
    frequences = obtener_frequencies(v, d)
    ajouter_frequencies(frequences)

res = associations(frequencies_m1, frequencies_d1, frequencies_m2, frequencies_d2)
intersections_1 = res["intersections_1"]
intersections_2 = res["intersections_2"]
points = res["points"]

for type_droites, droites in res["droites"].items():
    if type_droites == "droites_m1":
        c = "black"
    if type_droites == "droites_d1":
        c = "green"
    if type_droites == "droites_m2":
        c = "red"
    if type_droites == "droites_d2":
        c = "blue"
    for droite in droites :
        plt.plot([-100, 100], [droite["m"]*(-100) + droite["b"], \
            droite["m"]*100 + droite["b"]], c)

plt.scatter([point[0] for point in intersections_1], \
    [point[1] for point in intersections_1], \
    s = 40, c = "black", marker = "s", \
    label = str(len(intersections_1)) + " points triangle 1")
plt.scatter([point[0] for point in intersections_2], \
    [point[1] for point in intersections_2], \
    s = 40, c = "red", marker = "s", \
    label = str(len(intersections_2)) + " points triangle 2")

plt.scatter([point[0] for point in vrais_points], \
    [point[1] for point in vrais_points],

```

```
s = 30, c = "green", marker = "s")
plt.scatter([point[0] for point in points], [point[1] for point in points],
s = 20, c = "orange", marker = "s",
label = str(len(points)) + " points trouvés")

plt.legend()
plt.show()
```

Table des figures

1	Un radar uRAD	1
2	Tâche donnée à chaque personne du groupe	4
3	Diagramme de Gantt du S4	4
4	Diagramme de bête à corne du S4	6
5	Diagramme FAST du S4	7
6	Architecture fonctionnelle du S4	7
7	Architecture physique du S4	8
8	Localisation des coureurs dans un plan (O, x, y) avec deux radars	10
9	Trilateration	10
10	Illustration de la distorsion	11
11	Notre disposition finale	11
12	Visualisation des mesures sur la vidéo	12
13	Redis	13
14	Le réseau local	13
15	La modulation fréquentielle de nos radars	14
16	Apparition de cibles-fantômes	15
17	Utilisation d'un deuxième triangle pour supprimer les cibles-fantômes	16
18	Résolution en vitesse et nombre d'échantillons	17
19	La page principale	19
20	L'utilisateur indique à la vidéo la longueur du vecteur qu'il vient de tracer	19
21	Les mesures sont affichées dès que l'utilisateur a enregistré les positions des radars	20
22	Coureur rapide	20
23	Coureur lent	20
24	Test unitaire de notre algorithme à seuil adaptatif	22
25	Test unitaire de notre algorithme à seuil adaptatif	22
26	Notre test de validation fonctionnelle	23

Références

- [1] ALEXANDER WONG, MAHMOUD FAMUORI, M. J. S. F. L. B. C., AND CHUNG, J. Yolo nano: a highly compact you only look once convolutional neural network for object detection. *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing* (oct 2019). [17](#)
- [2] DALAL, N., AND TRIGGS, B. Histograms of oriented gradients for human detection. *International Conference on Computer Vision Pattern Recognition (CVPR '05), San Diego, United States* (jun 2005). [17](#)
- [3] JOSEPH REDMON, SANTOSH DIVVALA, R. G. A. F. You only look once: Unified, real-time object detection. *Conference: 2016 IEEE Conference on Computer Vision and Pattern Recognition* (jun 2016). [17](#)
- [4] MYKHAYLO ANDRILUKA, STEFAN ROTH, B. S. People-tracking-by-detection and people-detection-by-tracking. *2008 IEEE Conference on Computer Vision and Pattern Recognition, Anchorage, AK, USA* (aug 2008). [15](#), [17](#)
- [5] SCHNEIDER, N., AND GAVRILA, D. M. Pedestrian detection in video surveillance using fully convolutional yolo neural network. *Pattern Recognition: 35th German Conference, GCPR 2013, Saarbrücken, Germany* (sep 2013). [18](#)
- [6] VLADIMIR V. MOLCHANOV, BORIS V. VISHNYAKOV, Y. V. V. O. V. V., AND KNYAZ, V. V. Pedestrian detection in video surveillance using fully convolutional yolo neural network. *Proc. SPIE 10334, Automated Visual Inspection and Machine Vision II* (jun 2017). [17](#)
- [7] YOUN-SIK SON, HYUK-KEE SUNG, S. W. H. sensorsarticleautomotive frequency modulated continuous waveradar interference reduction using per-vehiclechirp sequences. www.mdpi.com (july 2018). [15](#)