



SORBONNE UNIVERSITÉ SCIENCES

PANDROIDE

Mise en place d'un système de tracking visuel et réalité augmentée pour robotique en essaim

*Antonin ARBERET 3407709
Jonathan MORENO 3530148*

Encadré par
Nicolas BREDECHE

Mai 2019

Table des matières

1	Introduction et contexte	2
1.1	Robotique en essaim, kilobots	2
1.2	Objectifs du projet	3
2	Présentation et comparaison des méthodes de tracking existante	4
2.1	MARGO : Massively Automated Real-time GUI for Object-tracking	4
2.2	SwisTrack	5
2.3	ARK : Augmented Reality for Kilobots	5
3	Mise en place d'ARK et de son environnement	7
3.1	Arène et matériel	7
3.2	Diagnostic et configuration de la machine	8
3.3	Simulation de l'environnement	8
3.3.1	Structure, dépendances et incompatibilités	8
3.3.2	Docker	9
3.3.3	Corrections des imports	10
3.4	Exécution, analyse et remarques	10
3.4.1	Nombre et type de caméras, format de l'arène	10
3.4.2	Calibration	11
3.4.3	Analyse critique du logiciel et possibilités d'amélioration	13
4	Modifications du nombre de caméras nécessaires	14
4.1	Couverture possible de l'arène avec une ou deux caméras	14
4.1.1	Deux caméras	14
4.1.2	Une caméra	14
4.2	Modification du code	15
5	Horizon du projet et conclusion	16

Chapitre 1

Introduction et contexte

La robotique en essaim est une branche de la robotique qui s'inspire des espèce animales vivant en collectivité capables d'accomplir collectivement des tâches complexes. Pour ce faire, elles adoptent généralement des comportements collectifs basés sur des interactions simples et locales alliant robustesse, flexibilité et scalabilité [1]. Les chercheurs dans ce domaine modélisent, étudient et imaginent des comportements de ce genre à l'aide de grands groupes de robots qu'on appelle essaims.

Du point de vue de l'observateur, l'étude de ces comportements collectifs peut se révéler laborieuse et une solution pour la rendre plus efficace est de mettre en place un système permettant de suivre les robots dans leur évolution.

Dans le cadre de ce projet nous avons été accueillis par Nicolas Bredèche à l'Institut des Systèmes Intelligents et de la Robotique (ISIR) pour travailler sur un système de tracking vidéo en réalité augmentée destiné à être utilisé avec l'essaim de robots du laboratoire.

Ce rapport et le code du Docker sont disponibles sur le dépôt suivant :

<https://github.com/AntoninARBERET/Pandroide-ARK>

Les autres dépôts de code liés au projet y sont référencés également.

1.1 Robotique en essaim, kilobots

L'ISIR possède un essaim de robots nommés kilobots [9] . La principale contrainte dans l'étude des essaims de robots est le coût unitaire du robot, limitant fatallement la taille de l'essaim en fonction du budget. C'est dans l'optique de pallier ce problème que les kilobots ont été conçus.

Les kilobots sont aussi simples que possible, ils n'embarquent que les fonctionnalités élémentaires nécessaires aux fonctionnement d'un essaim de robots : le déplacement et la communication entre robots. Grâce aux tiges métalliques faisant office de pattes et aux deux moteurs vibreurs qui, lorsqu'ils sont en fonctionnement, appliquent une force tangente aux disques formant la base du kilobot, ils ont la capacité de pivoter et de se déplacer en ligne droite sur une surface plane et lisse.

Leurs communications sont assurées par un émetteur récepteur infrarouge situé sous le robot, qui doit donc évoluer sur un surface réfléchissante pour communiquer avec les autres. Les communications ne se font qu'avec les proches voisins (environ 10 cm maximum). Ce capteur permet aussi de recevoir des messages d'un émetteur central, nommé Overhead Controller (OHC), situé au-dessus de leur environnement, c'est par celui-ci que l'utilisateur programme les robots.

Nous nommerons ?arène ? cet environnement spécifique nécessaire à l'utilisation des kilobots.

Grâce à leur simplicité les robots ne coûtent que 14\$ par unité, ce qui permet à des laboratoires de recherche d'accueillir des essaim de centaines, voir de milliers de kilobots.



FIGURE 1.1 – Un kilobot

1.2 Objectifs du projet

Initialement, les objectifs du projet étaient de mettre en place le système de tracking en réalité augmentée et d'y ajouter de nouvelles fonctionnalités, en particulier la gestion d'un nombre variable de caméras.

Chapitre 2

Présentation et comparaison des méthodes de tracking existante

Dans cette partie nous allons aborder la thématique du tracking visuel en évoquant notamment quelques logiciels et techniques déjà mis en place et fonctionnels. Mais avant cela il convient de donner une définition du sujet afin de cadrer la thématique.

Le tracking visuel consiste en la capture et le suivi d'objets en mouvement en utilisant la plupart du temps une caméra. Cette capture permet l'extraction d'informations des objets suivis, l'une des informations principales est leur déplacement, cette information nous intéressera particulièrement dans notre cas. Dans le but d'illustrer nos propos nous allons étudier quelques logiciels appliquant le tracking visuel. Nous commencerons par l'étude du Logiciel MARGO

2.1 MARGO : Massively Automated Real-time GUI for Object-tracking

MARGO [11] est un logiciel de tracking visuel développé par des chercheurs de l'université d'Harvard et de celle de Cambridge en partenariat avec l'Institut de Zoologie de l'Université de Regensburg. Il est passé open source depuis la fin du mois de mars 2019 et son environnement est développé en MATLAB. Celui-ci a été conçu pour l'étude de comportements d'animaux et plus précisément du mouvement de certaines espèces comme les mouches ou les bourdons.

Ce logiciel est capable, en théorie, d'effectuer un tracking sur un très grand nombre d'individus (l'ordre du millier), en pratique, les expériences publiées ne dépasse pas quelques centaines.

Il a la possibilité de suivre les animaux en temps réel ce qui est un véritable avantage pour les chercheurs étudiant les comportements et plus précisément leurs mouvements. Par exemple, il permet d'envoyer certains stimulus aux animaux et de voir leurs réactions instantanées et donc d'adapter les stimulus envoyés pour obtenir le comportement souhaité ou comprendre quels stimuli déclenchent tel ou tel comportement.

Deux modes principaux sont présents dans ce logiciel :

- Un mode de tracking individuel qui consiste en l'étude du comportement de plusieurs individus tous isolés les uns des autres. Chaque individu est ainsi disposé dans des boîtes circulaires de contours noirs et de fond translucide. Ces boîtes sont ensuite mises dans une plus grande boîte ayant un sol illuminé de lumière blanche. L'individu est ainsi repéré par un simple traitement d'image en recherchant la couleur de l'espèce étudiée sur le fond blanc. Avec ce mode, on peut ainsi identifier chacun des individus puis suivre leur comportement individuel.
- Un mode de tracking de groupe qui consiste lui en l'étude du comportement d'un groupe d'individu dans un environnement commun, par exemple un nid de bourdons. L'étude de tels groupes ne nous permet pas d'identifier chacun des individus en lui assignant un numéro mais assure le

tracking de chacun des individus anonymement. Cela démontre une robustesse dans le résultat fourni.

Les résultats sont fournis sur l'écran de l'utilisateur et sont représentés par l'image capturée sur laquelle chaque disque coloré représente un individu étudié. Ce système de tracking constitué d'une unique caméra est efficace pour son domaine d'étude. Cela est de plus possible avec un hardware accessible.

[margo article]

2.2 SwisTrack

SwisTrack [10] est aussi un logiciel de tracking visuel développé par des chercheurs de l'École Polytechnique Fédérale de Lausanne en Suisse. La dernière version est disponible depuis fin octobre 2008, ce logiciel est open source et a été développé en C++.

Celui-ci a été conçu pour le tracking visuel de systèmes multi-agents, comme par exemple une population de robots ou d'animaux. De même que pour MARGO, SwisTrack est un logiciel capable de fonctionner directement en temps réel mais également avec un enregistrement vidéo de la population à étudier.

Contrairement au logiciel précédent, la taille de la population étudiée ne peut excéder quelques dizaines d'individus. Les exemples vus ne dépassent pas une population de 25 individus.

Plusieurs exemples d'expériences ont été effectués, certaines d'entre elles nous intéressent particulièrement : au moins deux étudient des robots. L'une d'elle portait sur le tracking de plusieurs e-puck [5]. Un e-puck est un petit robot (70mm de diamètre par 50mm de hauteur) idéal pour l'enseignement et la recherche.

Afin de pouvoir identifier chaque robot individuellement, on place sur lui un certain motif noir et blanc. Ces motifs sont reconnus par un traitement d'image simple et associés à un identifiant. Par la suite, chaque robot est suivi à l'aide de la caméra capturant l'image. Le retour sur l'écran est constitué de l'image filmée à laquelle on ajoute un point coloré sur chaque robot, le rendu est assez similaire à celui de MARGO.

L'avantage est que l'on peut distinguer chaque robot individuellement et étudier le comportement précis de chacun à tout instant.

De plus, le système a été testé pour un groupe de cafards, et fonctionne tout aussi bien. Cependant la vitesse et le comportement des cafards ont posé quelques problèmes. Les cafards sont rapides et peuvent se monter dessus, on ne peut donc pas assurer une identification précise de chaque individu. Le taux d'erreur est assez faible mais lors de l'observation des résultats on peut remarquer que lorsque deux cafards se montent dessus leurs identifiants sont parfois échangés.

Un autre avantage de ce logiciel est la possibilité d'ajout d'une seconde caméra, qui augmente la zone d'étude possible. Mais certaines conditions sont requises, il faut que les deux images des caméras aient une zone d'overlap (superposition) assez grande.

En effet, la fusion d'image coûte cher en terme de ressource mémoire, c'est pourquoi dans ce cas, le traitement d'image ne peut plus être effectué en temps réel. Le logiciel ne fusionne pas les flux vidéo vers une unique représentation de l'arène mais maintient deux représentations de l'arène en évaluant la probabilité qu'un robot présent dans une zone commune aux deux caméras soit le même robot.

Comme pour MARGO, SwisTrack peut être exécuté sur des systèmes à performances raisonnables.

2.3 ARK : Augmented Reality for Kilobots

ARK [8] est le logiciel sur lequel porte notre projet, celui-ci a été développé par des chercheurs de l'Université de Sheffield. Disponible depuis 2017, il est open source comme les deux autres logiciels vus précédemment. Développé en C++, il a été spécialement conçu pour l'étude de comportements d'essaims de kilobots.

ARK est basé sur un système de caméra placé au-dessus de l'arène, capable de détecter chaque kilobot puis de suivre leur déplacement tout au long d'une expérience. Pour cela ARK utilise la biblio-

thèque de traitement d'image en temps réel d'Intel OpenCV [3], adaptée pour être supportée par la bibliothèque d'accélération de calculs par GPU de Nvidia CUDA [6]. Ainsi ARK est capable d'analyser en temps réel la position de chaque kilobot dans l'arène pour des essaims de plus de 200 robots.

De plus, ARK peut communiquer avec les kilobots via l'OHC (le contrôleur) en temps réel avec un ou tous les kilobots, ce qui lui permet de limiter les actions de ces derniers.

Ce logiciel répond parfaitement aux attentes de l'ISIR, dans un premier temps il identifie chacun des kilobots et lui assigne un identifiant unique. Cela nous permet de distinguer chacun des kilobots les uns par rapport aux autres et en cela d'augmenter leurs possibilités. En effet, un kilobot ne peut communiquer et localiser les autres kilobots mais à l'aide du contrôleur cela est dorénavant possible. Le logiciel connaît à chaque instant la position de chaque kilobot. Il peut donc communiquer avec chaque kilobot la position d'un de ses homologues. Ce processus multiplie les possibilités d'expérience imaginables avec les kilobots.

ARK a plusieurs avantages nous permettant de modéliser des situations d'étude différentes.

ARK peut par exemple simuler des éléments sur l'arène en réalité augmentée, simplement en limitant les actions des kilobots lorsqu'ils arrivent dans la zone de l'élément : ils peuvent être contraints de se diriger vers une zone d'intérêt, empêchés d'entrer dans un obstacle ou acquérir une information lorsqu'ils se trouvent dans une zone particulière. On peut donc leur assigner la tâche de collecter une ressource simulée sur un point de l'arène à placer ensuite dans une zone de collecte puis étudier leur comportement.

ARK permet aussi de faciliter le mouvement des robots les uns par rapport aux autres. Le système de communication des kilobots ne leur permet pas de connaître la position de leur voisin mais uniquement la distance jusqu'à eux. ARK est donc le seul des logiciels présentés à utiliser quatre caméras.

Dans un premier temps, ARK effectue une calibration des caméras puis il fusionne les images afin d'obtenir une vision complète de l'arène où se situent les kilobots. Sur cette image, il détecte les kilobots, la couleur de leur diode et assigne à chacun un identifiant unique. Le logiciel pouvant être exécuté en temps réel, les variables de l'environnement peuvent être modifiées en direct.

De plus, ARK réduit les temps d'expérience, en automatisant plusieurs étapes fastidieuses comme le positionnement initial des robots, la calibration de leurs moteurs et l'assignation d'identifiants et il permet également d'enregistrer les données expérimentales.

Chapitre 3

Mise en place d'ARK et de son environnement

3.1 Arène et matériel

L'arène de l'ISIR est une arène rectangulaire de 152 cm sur 72 cm dont la surface est en velleda blanc pour permettre les communications.

Elle est équipée de supports transversaux suspendus au-dessus de sa surface pour y fixer les caméras et l'OHC. La hauteur des supports peut être fixée jusqu'à 120 cm.

Les quatre caméras sont des Logitech c920 [4] qui capturent en 1920x1080 pixels.

L'essaim qui évolue sur cette arène est composé d'une centaine de kilobots.

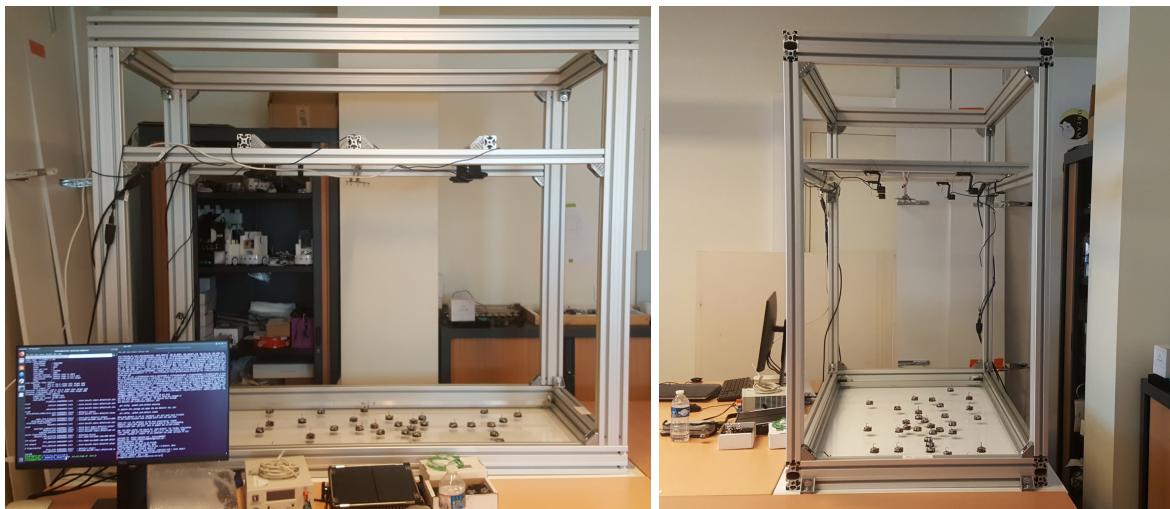


FIGURE 3.1 – L'arène de l'ISIR

3.2 Diagnostic et configuration de la machine

En premier lieu, à notre arrivée à l'ISIR, nous avons eu à faire l'état des lieux des machines disponibles pour en dégager une capable de faire tourner ARK. Plusieurs contraintes matérielles sont liées à ARK, en particulier la carte graphique qui doit être une Nvidia suffisamment performante pour permettre à CUDA d'accélérer OpenCV en temps réel pendant l'expérience, beaucoup de mémoire vive doit ainsi être disponible.

Malheureusement aucune des machines présentes à ce moment-là ne satisfaisait ces contraintes, il a donc été décidé d'en commander une nouvelle. La configuration de cette machine est :

- OS : Ubuntu 18.04
- CPU : Intel Xeon W-2155 @ 3.3GHz
- GPU : 1x Nvidia Quadro P400 (RAM : 2Go), 1x GeForce RTX 2080 Ti (RAM : 10Go)
- RAM : 32Go de RAM
- SSD : 250Go

Le premier mois du projet a donc été consacré à la documentation, l'étude du code et la découverte et la prise en main des kilobots en attendant d'avoir accès à un environnement adapté à l'installation de ARK.

3.3 Simulation de l'environnement

À la réception de la machine, nous avons plongé dans la partie de ce projet qui, contre toute attente, s'est révélée être la plus fastidieuse : la mise en place de l'environnement et l'installation d'ARK.

3.3.1 Structure, dépendances et incompatibilités

ARK est composé de deux parties distinctes : le logiciel et le programme qui effectue la calibration des caméras.

Ces deux composantes utilisent OpenCV pour faire l'acquisition des flux vidéo et effectuer les traitements d'image. Ces opérations étant coûteuses, il est difficilement possible de les effectuer en temps réel. C'est pourquoi la version d'OpenCV utilisée est spécifiquement adaptée pour utiliser CUDA 8, une bibliothèque d'accélération de calcul par GPU proposée par Nvidia.

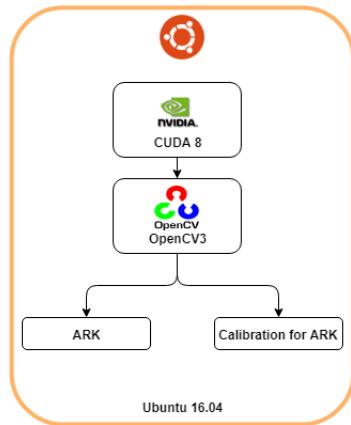


FIGURE 3.2 – Structure de l'environnement d'ARK à Sheffield

La première problématique que nous avons rencontrée est l'incompatibilité entre notre système d'exploitation et ces bibliothèques : la nouvelle machine était sous Ubuntu 18.04 qui n'est pas compatible avec CUDA 8, et les versions suivantes de CUDA ne sont pas compatibles avec l'adaptation d'OpenCV.

Nous aurions alors simplement pu passer sous Ubuntu 16.04, mais ces premiers problèmes de compatibilité nous ont poussés vers une solution plus robuste qui permettrait à d'autres de ne pas se heurter à des problématiques similaires : simuler l'environnement de ARK avec Docker.

3.3.2 Docker

Docker [2] est un logiciel qui permet d'encapsuler une application avec son environnement et toutes ses dépendances dans une image. Cette image est conservée localement mais peut être facilement transférée à l'aide d'un fichier Dockerfile qui comporte toutes les instructions à exécuter pour re-créer l'environnement. L'image peut alors être reconstruite et exécutée dans un conteneur Docker sur n'importe quelle machine à la manière d'une machine virtuelle. Dans une optique de robustesse au changement de hardware ou à l'utilisation par d'autres laboratoires de recherche, Docker nous permet donc d'apporter la garantie de pouvoir exécuter ARK rapidement dans les mêmes conditions que celles de l'ISIR.

Nous avons donc décidé de mettre au point une image de l'environnement d'ARK au complet, et nous avons dû faire face à plusieurs problématiques pour y arriver.

À la création d'un environnement Docker, on part d'une version initiale du système d'exploitation. Dans notre cas, nous avons choisi de partir d'une version d'Ubuntu 16.04 déjà équipée de CUDA 8 proposée par Nvidia spécialement pour Docker. À ce moment-là, le système ne contient que le strict minimum et n'a que très peu de permissions sur l'utilisation du hardware sur lequel il est réellement exécuté.

Pour assurer les performances du système, il a ensuite fallu faire en sorte que le conteneur Docker puisse accéder à la puissance GPU de la machine. Nous utilisons pour cela Nvidia Docker [7] développé par Nvidia pour permettre l'exécution d'applications accélérées par sa technologie dans des conteneurs Docker. C'est lui qui assure le lien entre le conteneur et la carte graphique. Il est donc nécessaire de l'installer à côté de Docker pour faire tourner le conteneur.

ARK étant une application dont la force repose notamment sur son interface graphique, qui permet à l'utilisateur de suivre l'expérience en temps réel, il faut aussi passer l'accès à l'affichage au conteneur. On passe donc au Docker un paramètre display avec le group id et user id de l'utilisateur et la permission est accordée à la construction de l'image.

Enfin les accès aux caméras et à l'Overhead Controller sont passés en paramètre à l'exécution.

Toutes ces contraintes obligent l'utilisateur à passer certains paramètres à Docker pour la construction et l'exécution de l'image. Nous les avons donc encapsulés dans des scripts bash détaillés dans la documentation utilisateur.

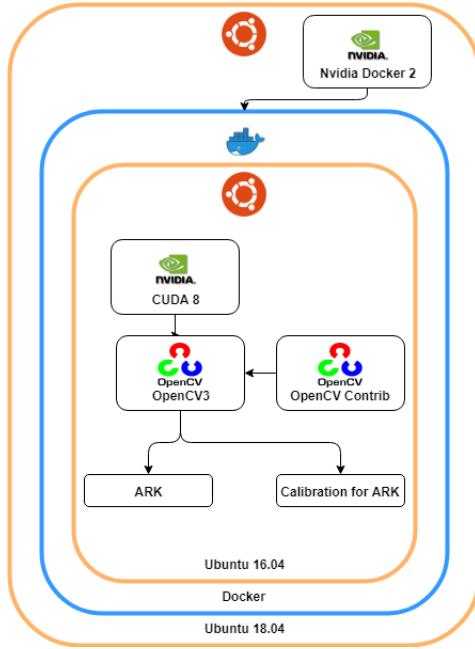


FIGURE 3.3 – Structure de l’environnement d’ARK à l’ISIR

3.3.3 Corrections des imports

À ce stade nous avons reproduit exactement l’environnement utilisé par les chercheurs de Sheffield. Cependant nous avons rencontré de nouveaux problèmes de compilation avec ARK et avec le logiciel de calibration. Après recherche, il s'est avéré que certains modules d'OpenCV utilisés par ARK n'existaient pas dans la version d'OpenCV recommandée. Nous avons donc contacté le chercheur qui mène le projet ARK. Il nous a répondu qu'une version d'OpenCV 2 était installée à côté d'OpenCV 3, et que certains modules en étaient probablement importés mais pas utilisés.

Des erreurs persistaient après avoir retiré les imports inutiles, et nous avons compris qu'OpenCV 2 était utilisé par défaut pour certaines fonctionnalités ayant été retirées de la version principale d'OpenCV 3. Entre autres, les algorithmes de description d'image SIFT et SURF ont été placés dans le paquet externe d'OpenCV : contrib. Enfin, après l'ajout de ce paquet, nous étions en mesure d'exécuter sans erreurs ARK et son logiciel de calibration.

La simulation de l'environnement est finalement la majeure partie de notre travail sur machine. La découverte de Docker, nos compétences limitées en administration système ainsi que la nécessité de tester sur la machine de l'Université ont rendu cette tâche extrêmement chronophage.

3.4 Exécution, analyse et remarques

Nous n'en sommes arrivés au point de pouvoir exécuter le logiciel qu'une semaine avant la date de rendu de ce rapport. Les résultats présentés ici sont les derniers que nous avons obtenus au laboratoire. La version de ARK est pratiquement la version originale car notre version n'était pas fonctionnelle, elle n'est donc pas adaptée à l'arène de l'ISIR. Ces résultats peuvent donc sans doute être améliorés mais nous avons été pris par le temps.

3.4.1 Nombre et type de caméras, format de l'arène

Comme précisé précédemment, ARK ne fonctionne qu'avec quatre caméras. De plus, leur résolution n'est pas modifiable dans le logiciel, or Sheffield utilise des caméras format 4 :3, tandis que les

nôtres sont en 16 :9. Nous avons modifié l'appel à OpenCV qui précise la définition de l'acquisition, sans succès. Le code est très peu commenté, il est très probable que nous ayons manqué une autre instruction impactant ce paramètre.

De plus chaque caméra filme normalement une zone carrée représentant un quart de l'arène. Le logiciel ne peut donc fonctionner parfaitement qu'avec des arènes carrées.

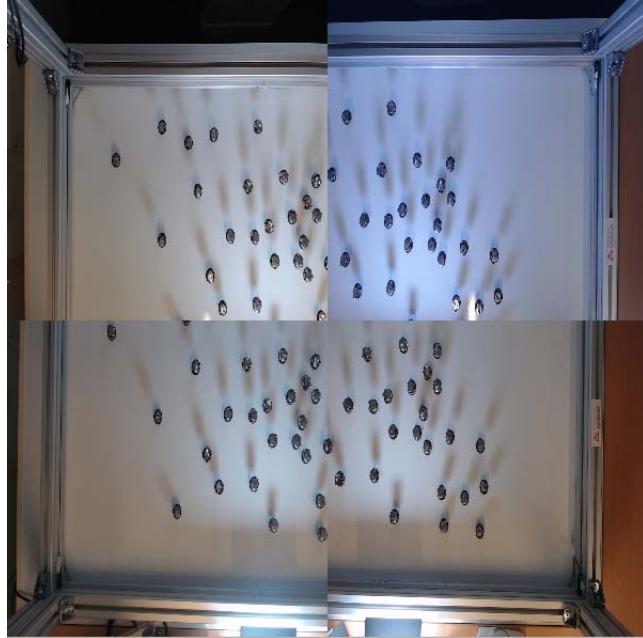


FIGURE 3.4 – Capture de l'arène avant la calibration

3.4.2 Calibration

Le logiciel de calibration est basé sur SURF et la détection de points d'intérêt similaires dans les différentes images. Deux paramètres sont modifiables par l'utilisateur à la main et les configurations dans lesquelles les points d'intérêt se recoupent sont rares.

Cette méthode nécessite alors une zone de chevauchement entre les images qu'on appelle overlap. Nous avons donc testé deux méthodes : un grand overlap en ne filmant que l'arène ou un petit overlap en filmant autour de l'arène également.

Sur les images suivantes les points colorés sont des points d'intérêts. Les bleu foncé sont solitaires et chaque paire parmi les quatre images a une couleur associée. Ainsi, un point non bleu est commun à deux photos.

Petit overlap

Dans cette configuration l'overlap se rapproche de celui utilisé à Sheffield par les créateurs d'ARK. En revanche nous sommes contraints de filmer la zone autour de l'arène ce qui pose des problèmes. En particulier dans l'image ci-dessous on remarque que le détecteur associe des points rouges complètement différents au niveau des angles de l'arène. En calibrant avec cet overlap nous n'avons eu aucun résultat convenable.

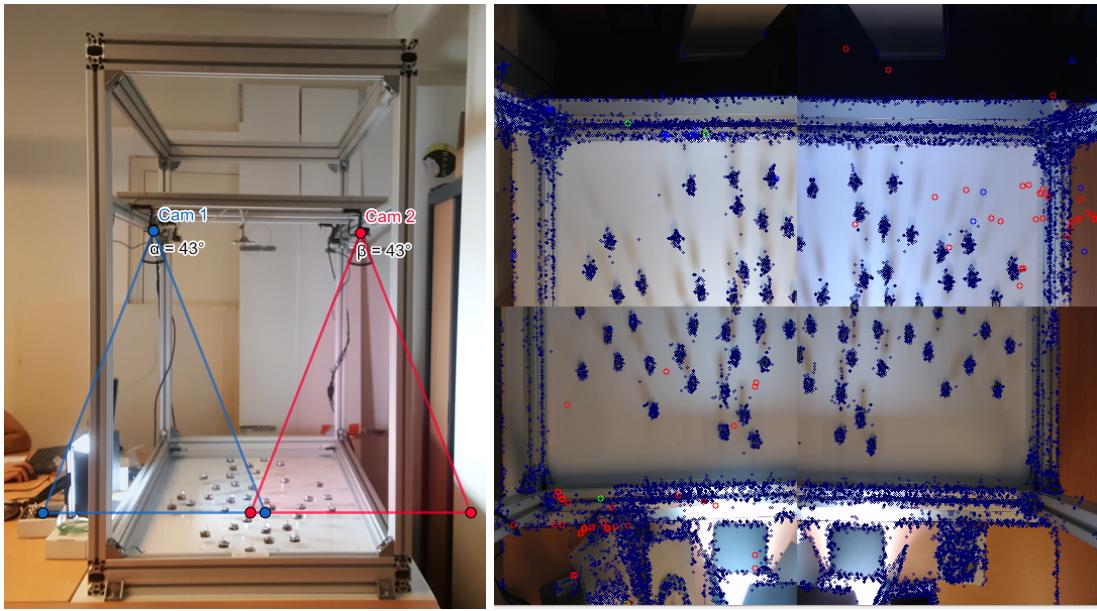


FIGURE 3.5 – Détection des points communs à la calibration avec petit overlap

Grand overlap

En grand overlap les caméras filment une grande surface de l'arène en commun. On peut /alors limiter l'acquisition aux bords de l'arène. Nous avons réussi à obtenir dans cette configuration une calibration qui ressemble à une arène.

Suite à l'étape de jointure sur les points communs, l'image de l'arène passe par une étape "Squared" qui rend carrée l'arène dont on sélectionne les coins, ce qui constitue une nouvelle contrainte sur la forme possible de l'arène.



FIGURE 3.6 – Détection des points communs à la calibration avec grand overlap, puis fusion des images

Malheureusement, une fois la calibration donnée à ARK, le résultat n'a plus de sens. Cela nous a permis tout de même de lancer ARK qui ne peut pas faire de capture sans le fichier de calibration, mais nous ne savons pas actuellement d'où vient ce problème. L'image ci-dessous à droite est la capture normalement calibrée faite par ARK, qui tente ensuite d'y détecter des kilobots.

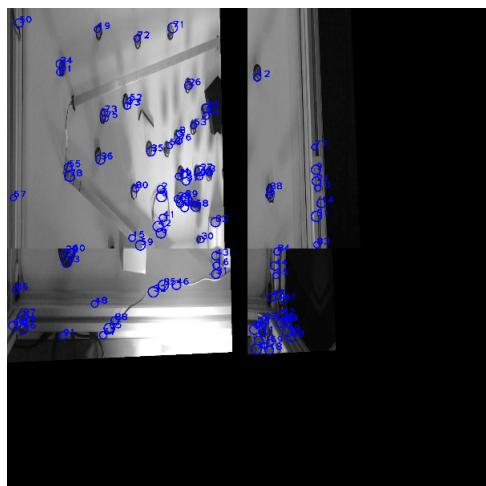


FIGURE 3.7 – Image renvoyée par ARK à l'identification des kilobots

3.4.3 Analyse critique du logiciel et possibilités d'amélioration

ARK a été développé pour fonctionner dans l'environnement du laboratoire de Sheffield et il nous semble qu'il n'a pas été conçu dans l'optique d'une utilisation dans des environnements différents : beaucoup d'éléments sont codés en dur, presque rien n'est modifiable depuis le logiciel.

Dans le cadre d'une amélioration du logiciel, beaucoup de pistes sont à exploiter pour lui faire gagner en robustesse : la possibilité de modifier le nombre de caméras [cf partie] évidemment, mais aussi un grand nombre de choses sur lesquelles l'utilisateur devrait avoir la main comme -la définition et le format des caméras

-la forme de l'arène, on pourrait ainsi masquer la capture et éliminer les contours de l'arène directement dans le logiciel

-une automatisation du paramétrage de la détection qui permettrait de trouver les configurations intéressantes rapidement

Ces modifications traversent toutes les parties du logiciel de l'acquisition des images à l'interface utilisateur en passant par la calibration. Elles sont toutes interdépendantes, et demandent un travail conséquent sans lequel il ne sera sûrement pas possible de travailler sur l'arène de l'ISIR.

Chapitre 4

Modifications du nombre de caméras nécessaires

4.1 Couverture possible de l'arène avec une ou deux caméras

Nous avons effectué quelques mesures pour vérifier, avec les caméras actuelles, quelles nouvelles configurations nous pouvions proposer. Les caméras de l'ISIR capture selon un angle horizontal d'environ 70° et un angle vertical d'environ 43° .

4.1.1 Deux caméras

Il est possible de couvrir l'arène avec deux caméras. Cela implique de fixer la hauteur au minimum à 95 cm pour que les caméras puissent couvrir l'arène dans toute sa largeur, et donc un overlap relativement conséquent sur la longueur.

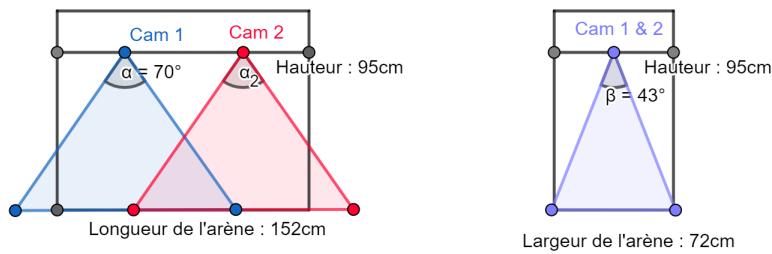


FIGURE 4.1 – Couverture de l'arène de l'ISIR avec deux caméras

4.1.2 Une caméra

On arrive à couvrir presque parfaitement l'arène avec une unique caméra placée à environ 115 cm au-dessus de l'arène. Cependant, il est possible que le fort angle avec les kilobots les plus excentrés provoque des erreurs d'identifications.

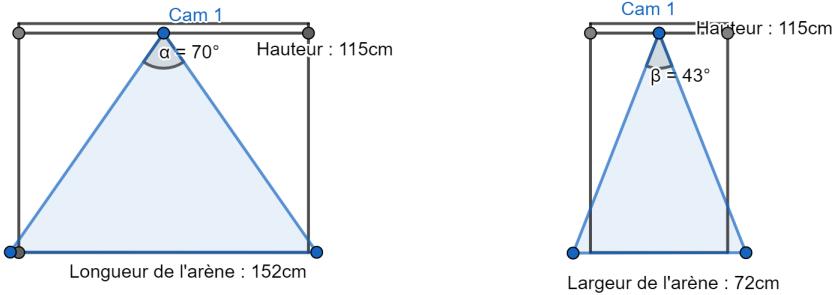


FIGURE 4.2 – Couverture de l'arène de l'ISIR avec une caméra

4.2 Modification du code

Afin de modifier le code pour diminuer le nombre de caméras, il convenait de modifier les deux parties du logiciel. Nous avons commencé les changements en parallèle du travail effectué sur l'environnement pour prendre de l'avance mais sans pouvoir tester notre version.

Lors de ces changements, nous nous sommes heurtés à plusieurs barrières. En effet, le code n'étant pas ou peu commenté et rempli d'appels à des bibliothèques que nous ne maîtrisons pas, il nous était difficile de comprendre la façon dont le créateur gérait les caméras. Cette partie du code est majoritairement codée en dur, avec des boucles de 1 à 4 disséminées un peu partout dans le code.

Malheureusement, nous n'avons pu exécuter le logiciel que très tardivement, et nous avons découvert à ce moment l'ampleur de ce qu'il faudrait modifier dans ARK pour réduire le nombre de caméras et gérer une arène rectangulaire. Nous n'avons donc pas à ce jour de version fonctionnelle à proposer pour répondre à cette problématique.

Chapitre 5

Horizon du projet et conclusion

À l'issue de ce projet, nous avons mis au point un environnement ARK encapsulé dans une image Docker complet et dans lequel les différentes erreurs liées à l'importation ont été corrigées. Nous n'avons pas eu le temps d'aller jusqu'à une modification fonctionnelle du logiciel, mais nous espérons avoir fourni des analyses qui permettront que cette dernière puisse être faite de façon robuste et efficace.

Pour nous, cette expérience fut totalement différente de celle que nous attendions, mais nous avons eu l'occasion d'apprendre beaucoup sur l'administration système et l'utilisation de conteneur Docker. En parallèle nous avons découvert une petite facette de la robotique en essaim, ce qui fut très intéressant.

Bibliographie

- [1] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics : a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1) :1–41, March 2013.
- [2] Docker. Docker. <https://www.docker.com/>. En ligne ; consulte le 19 Mai 2019.
- [3] Intel. OpenCV : Open Source Computer Vision Library. <https://opencv.org/>. En ligne ; consulte le 19 Mai 2019.
- [4] Logitech. c920 specifications. https://support.logitech.com/en_us/product/hd-pro-webcam-c920/specs. En ligne ; consulte le 19 Mai 2019.
- [5] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stephane Magnenat, Jean-Christophe Zufferey, Dario Floreano, and Alcherio Martinoli. The e-puck, a Robot Designed for Education in Engineering. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 1(1) :59–65, 2009.
- [6] Nvidia. CUDA : Compute Unified Device Architecture. <https://www.nvidia.fr/object/cuda-parallel-computing-fr.html>. En ligne ; consulte le 19 Mai 2019.
- [7] Nvidia. Nvidia Docker : Build and run Docker containers leveraging NVIDIA GPUs. <https://github.com/NVIDIA/nvidia-docker>. En ligne ; consulte le 19 Mai 2019.
- [8] A. Reina, A. J. Cope, E. Nikolaidis, J. A. R. Marshall, and C. Sabo. ARK : Augmented Reality for Kilobots. *IEEE Robotics and Automation Letters*, 2(3) :1755–1761, July 2017.
- [9] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot : A low cost scalable robot system for collective behaviors. In *2012 IEEE International Conference on Robotics and Automation*, pages 3293–3298, May 2012.
- [10] Lochmatter Thomas, Roduit Pierre, Cianci Chris, Correll Nikolaus, Jacot Jacques, and Martinoli Alcherio. SwisTrack - A Flexible Open Source Tracking Software for Multi-Agent Systems - IEEE Conference Publication.
- [11] Werkhoven Zach, Rohrsen Christian, Qin Chuan, Brembs Bjorn, and de Bivort Benjamin. MARGO (Massively Automated Real-time GUI for Object-tracking), a platform for high-throughput ethology | bioRxiv.