Utilisation de bibliothèque de vision et de perception pour une utilisation dans le domaine de la cobotique

UE 5.3 Rapport de Stage

FISE 2023

1^{er} **octobre 2022**

Simon Gervaise - simon.gervaise@ensta-bretagne.org

Remerciements

Je tenais à remercier David DANEY, mon tuteur, qui m'a accompagné pendant ce stage. Je tenais également à remercier entre autres Vincent PADOIS, Nicolas TORRES et Benjamin CAMBLOR qui ont pris de leur temps pour m'aider et apporter leur expertise.

Je remercie enfin les membres d'AUCTUS, qui ont été d'une sympathie et d'un accueil franc.

1 Résumé

La cobotique, contraction de robotique collaborative est une branche émergeante qui considère les interactions entre les robots et les humains. Un enjeu important est dans la perception de l'environnement et des humains en interaction. La vision est sans doute le domaine le plus avancé et le plus intéressant pour remplir les tâches de détection.

Bien que des solutions existent, celles-ci demandent toujours des ajustements pour répondre au mieux aux besoins spécifiques de chaque situation. Il est donc nécessaire d'identifier dans un premier temps d'identifier les différentes solutions les plus à même de répondres à nos besoins, de les mettre en oeuvre et de les comparer pour ensuite ajouter quelques modifications.

Une fois ces outils établis et opérationnels, il est important de pouvoir les partager de manière simple et rapide.

Abstract

Cobotics, a contraction of collaborative robotics, is an emerging field that considers the interactions between robots and humans. An important issue is the perception of the environment and of humans in interaction. Vision is probably the most advanced and interesting field to fulfil sensing tasks.

Although solutions exist, they still require adjustments to best meet the specific needs of each situation. It is therefore necessary to first identify the different solutions that are most likely to meet our needs, implement and compare them and then add some modifications.

Once these tools are established and operational, it is important to be able to share them in a simple and quick way.

Mot-clés

Cobotique, vision, détection, apprentissage automatique, librairie, docker

Keywords

Cobotic, vision, detection, machine learning, library, docker

Table des matières

Introduction

Dans le cadre de mes études d'ingénieur à l'ENSTA Bretagne, plusieurs stages sont inscrits dans le cursus. Au rythme d'un par année, ceux-ci ont pour but de nous faire découvrir le monde de l'ingénierie, que ce soit en entreprise ou en laboratoire mais aussi des acteurs du domaine de l'ingénierie avec qui nous pourrions être amenés à travailler. Ils nous servent également à utiliser ce que nous avons appris pendant l'année d'étude et d'étoffer notre savoir.

Pour ma deuxième année, j'ai donc réalisé un stage de quatre mois en tant qu'assistant ingénieur, afin de comprendre les enjeux et les problématiques auxquels doit faire face un ingénieur ou un chercheur. Cela permet aussi de mettre en perspective ce que nous avons appris à l'ENSTA Bretagne et de découvrir des exemples d'utilisations généralement insoupçonnés.

2 Introduction du stage

2.1 Contexte

L'INRIA (Institut National de Recherche en Informatique et en Automatique) est un centre de recherche basé sur plusieurs villes françaises dont Bordeaux. L'équipe AUCTUS, l'une des nombreuses présentes sur le site à Bordeaux, est en collaboration avec l'ENSC (Ecole Nationale Supérieure de Cognitique). Cette équipe travaille sur la cobotique, un domaine récent qui étudie la collaboration entre un humain et un robot.

AUCTUS est notamment motivée dans l'utilisation de cobot dans un secteur industriel en ayant des partenariats avec, entre autres, Stellantis, Airbus, Farm3 ou Solvay. L'équipe se compose essentiellement de doctorants qui étudient plusieurs branches de la cobotique et plus généralement de la robotique, comme la biomécanique, la cognitique ou encore la vision.

2.2 Objectifs du stage

2.2.1 Etat de l'art des librairies dans la vision et la perception

Comme dit plus haut, la cobotique est un domaine de la robotique qui met au centre la collaboration entre l'homme et le robot, il est donc nécessaire pour le robot de percevoir son environnement ainsi que l'opérateur.

La vision est un domaine développé, accessible et, dans de nombreux cas, suffisant pour l'utilisation des cobots, c'est pourquoi on se tourne généralement vers ces technologies.

L'utilisation de librairies dans la programmation se révèle précieuse afin d'utiliser les outils de vision et de réutiliser des travaux déjà existants. Il existe cependant un grand nombre de librairies, certaines répondant parfois aux mêmes besoin, avec des différences de performances. Il est donc important dans un premier temps de faire un état de l'art des librairies, d'analyser les besoins afin de choisir la librairie la plus performante pour l'adapter aux besoins.

2.2.2 Sélection et test des librairies

Il est important de savoir sur quels critères sélectionner les libraries afin de faire un premier écrémage. Celles qui semblent ne pas convenir aux problématiques et aux besoins sont rapidement écartées, ensuite celles restantes répondant aux mêmes besoins sont comparées.

Il est rare qu'une librairie soit, de manière absolue, meilleure qu'une autre. Il faut donc d'analyser les besoins afin de les prioriser et d'aider à la sélection des librairies.

2.2.3 Modification des librairies et transmission des programmes

Les besoins étant uniques selon les situations, il est commun de modifier les librairies afin de les rendre plus performantes et efficaces.

Bien que les librairies permettent un gain de temps significatif plutôt que de tout programmer soi-même, elles ne sont pas exemptes de complications.

Alors que Windows ou MacOS sont des OS (Operating System) largement utilisés sur des ordinateurs personnels, ceux-ci se font bien plus rares dans la robotique. Il est

en effet préféré l'utilisation d'OS Linux, en particulier Ubuntu. Cela permet une plus grande liberté dans son utilisation, comme de pouvoir modifier en profondeur l'ordinateur. Cela peut s'avérer nécessaire sur des robots que l'on veut configurer mais également sur des ordinateurs communiquant avec des robots. En revanche la liberté qu'apportent des OS comme Ubuntu prennent la place sur l'uniformité d'utilisation, ainsi, les librairies utilisées dans les programmes peuvent être sources d'erreurs lors de la compilation et peuvent compromettre le bon fonctionnement du programme.

Il est donc nécessaire de transmettre des librairies de la manière la plus "propre" possible afin que, quelque soit la configuration de l'ordinateur et de l'OS Ubuntu, il n'y ait pas de soucis à faire fonctionner le programme.

Enfin, ces librairies sont amenées à être integrées dans ROS, une interface qui permet la communication entre des robots et des ordinateurs.

3 Etat de l'art

3.1 Etude des besoins

Afin de comprendre les enjeux du stage, il a été dans un premier temps important de discuter avec l'ensemble du personnel du laboratoire afin de discuter des besoins de chacun, de ce qui se faisait déjà et des besoins futurs.

Les pré-requis étaient de pouvoir détecter un humain et des objets, ce sont des besoins essentiels dans la cobotique.

Un des doctorants, Benjamin CAMBLOR, travaillait d'un point de vue cognitique sur l'interaction homme-robot. Pour cela, le robot devait être capable de reconnaître des legos et en faire une pile. De son côté, l'humain devait procéder à un jeu des tours d'Hanoï tout en s'assurant que le travail du robot était bien effectué. Il fallait donc qu'une caméra filme l'expérience et puisse détecter où le sujet portait son attention.

Les autres travaux étaient moins sujets à des demandes aussi précises, ainsi il a fallu de mon côté, anticiper des besoins potentiels. J'ai donc regardé ce que les différentes librairies de vision proposaient comme solutions de manière très large, sans me soucier si cela pouvait servir ou non à l'équipe du laboratoire.

3.2 Comparaison de librairie répondant à ces besoins

C'est donc grâce à ces informations que j'ai pu commencer mes recherches sur les nombreuses librairies de vision. Mon tuteur m'a dans un premier temps recommandé de regarder la librairie OpenPose, l'une des plus répandues dans la robotique. C'est à partir de cette librairie que mes recherches ont commencé, j'ai rapidement noté d'autres librairies utilisées et ai constitué une liste assez fournie de librairies dans la vision.

Je n'ai dans un premier temps installé aucune d'entre elles, cela m'aurait pris bien trop de temps, en raison des incompatibilités possibles de configurations entre la librairie et l'OS utilisé. Je me suis donc référé au site en question, à des vidéos de démonstration et des avis sur internet pour me faire une première impression. J'ai ainsi pu établir plusieurs critères qui me semblait importants et j'ai donc crée un tableur pour comparer facilement toutes ces librairies. Pour plus de visibilité, j'ai séparé ce tableur en plusieurs autres.

Tout d'abord, l'information la plus accessible sur ces librairies est les solutions qu'elles proposent, ainsi il a été plutôt aisé de pouvoir les répertorier (figure 1).

Besoins/Librairies	Detection Mains	Detection Pose 2D	Detection Pose 3D	Detecion Visage	Detection expression	Calcul de Distance	Detection Collision	Position Objet	Orientation Objet	Couleur Objet	Nature Objet
OpenPose	X	X	X	X				X			
OpenNI	X	X		X							
OpenFace				X	X						
MediaPipe	X	X		X		X		X	X		X
AlphaPose	X	X									
MoveNet	X	X									
Flexible Collision						X	X				
OpenCV									X	X	
MocapNET		X	X	X							
Localizer									X		
MonoCon									X		
RoboFlow									X	X	X
TensorFlow								X	X		X
VIM								X	X		
	Tres bien documentee										
	Bien documentee										
	Moyennement documentee										
	Peu ou pas documentee										

FIGURE 1 : Solutions proposées par les librairies

On remarque dans un premier temps que de nombreuses solutions sont offertes par ces différentes librairies. Les détections de pose (partie du corps), de visage et de mains sont courantes, il va donc être aisé de pouvoir faire un choix qui va correspondre au mieux aux besoins de l'équipe. De plus, on comprend rapidement que ces technologies seront amenées à être utilisées largement, il est vital pour un cobot de pouvoir détecter le sujet pour le bien de l'expérience et éviter des accidents. On remarque également que les librairies qui proposent ces solutions de détection de pose, de visage et de main sont globalement très bien documentées. Cela est très intéressant dans la mesure où, afin d'adapter la librairie à une utilisation spécifique, cela sera bien plus simple afin de comprendre rapidement les codes et d'y apporter les modifications.

Au delà des solutions proposées par les librairies, il est primordial de pouvoir les comparer, notamment grâce à leur performance, selon plusieurs critères (figure 2).

Performances	Rapidite dexecution	Precision	Adaptibilite	Installation	Accessibilite	Robustesse	Portabilite
OpenPose	+	+	0	0		++	+
OpenNI					0		++
OpenFace	+	+	+		++		
MediaPipe	++	++	+	0	0	+	
AlphaPose	++	++	++	0		+	
MoveNet	++	+					
Flexible Collision							
OpenCV	++	+	0	++	+	++	+
MocapNET		++		+			
Localizer							
MonoCon	--			+	0		
RoboFlow	--	0	++	++	--	++	
TensorFlow	--	0	+	+	0	+	+
VIM	++	0	+	++	+		++

	Excellente
	Bonne
	Moyenne
	Mauvaise

FIGURE 2 : Performance des librairies

J'ai donc défini plusieurs critères qui me semblaient important, tout d'abord la

rapidité d'exécution. Les programmes doivent être rapides voire instantanés à se lancer afin d'éviter une mise en place de l'expérience trop longue.

La précision était sans doute le critère le plus déterminant. Pour mener à bien l'expérience, celle-ci se doit d'être précise, d'autant plus lorsqu'un humain est en jeu. Une imprécision peut mener à des accidents et cela doit être absolument evité.

L'adaptabilité est également importante, cela permet de savoir si les solutions peuvent être performantes dans des situations dans lesquelles elles ne sont pas censées fonctionner. Cela est un point important quand il s'agit d'adapter un programme pour un cas spécifique.

Un autre critère était la facilité d'installation. C'est notamment l'une des raisons de ce stage, l'installation de ces librairies est généralement fastidieux et peut se révéler long si les incompatibilités entre la librairie et la configuration de l'ordinateur et de l'OS sont trop nombreuses. De plus, au vu de la quantité de librairies disponibles, il n'était pas possible de toutes les installer afin de les comparer en profondeur, ainsi une librairie trop longue et problématique à installer est rapidement mise de côté.

L'accessibilité est la facilité à avoir accès aux codes, à les comprendre et à les modifier. Cela est nécessaire pour le travail que j'effectue mais également pour les modifications qui se feraient une fois mon stage terminé. Il est important que les modifications dans le code se fassent facilement et rapidement.

La robustesse est la réaction aux situations qui lui sont inconnues et qui pourraient provoquer des erreurs dans le programme. Si cela se passe pendant l'expérience, celle-ci se retrouve totalement inutile. Il est donc important que le programme couvre un maximum de possibilités afin d'éviter les erreurs potentielles dans le programme.

Enfin, la portabilité est la facilité à ce que le programme fonctionne quelque soit l'ordinateur. Bien que cela fasse parti de mon travail que les programmes aient une bonne portabilité, cela peut éviter un travail fastidieux.

Tous ces critères n'ont pas la même importance et le même poids lors du choix des librairies, on peut, par exemple, rapidement comprendre que la portabilité ou la robustesse auront moins de poids que la précision ou la rapidité d'exécution. Ces critères de performances, au-delà des solutions proposées par les librairies, ont été déterminant pour choisir.

On peut enfin noter que certaines librairies ont plusieurs, voire toutes leurs cases vierges. Cela indique qu'aucune information à ce sujet n'a été trouvé, ce qui devient assez problématique. Les librairies *Localizer* et *Flexible Collision* ont été rapidement écarté pour ces raisons et parce que les solutions proposées étaient présentes chez d'autres librairies ou bien pas forcément très utiles.

A cela s'ajoutait d'autres critères, certes moins important, mais pouvant être déterminant si deux librairies se valaient jusqu'ici. Ces librairies ont besoin d'une caméra pour fonctionner et chacune utilise une technologie bien précise de caméra (figure 3)

On remarque immédiatement que l'écrasante majorité des librairies fonctionnent avec une simple webcam ou une vidéo, ce qui est fort arrangeant. Ce sont des méthodes rapides à mettre en place et peu lourde dans le code. Ainsi, le choix des librairies s'est peu porté sur le type de caméra utilisée.

Librairies/Camera	Stereo	Flir Camera	WebCam	Video	Kinect	LIDAR
OpenPose	X	X	X	X		
OpenNi					X	
OpenFace			X	X		
MediaPipe			X	X	X	
AlphaPose			X	X		
MoveNet			X	X		
Flexible Collision			X	X		X
OpenCV			X	X		
MocapNET			X	X		
Localizer			X	X		
MonoCon			X	X		
VIM			X	X	X	

	Disponible
	Potentiellement disponible
	Indisponible

FIGURE 3 : Compatibilité avec les caméras

Il y a néanmoins eu une réflexion sur les caméras qui permettent une vision en 3D, c'est le cas pour la caméra stéréo, une caméra constitué de deux objectifs qui à partir, de deux images permet de reconstituer un environnement 3D et de calculer la profondeur (comme le font nos yeux) ; la Kinect, caméra de la Xbox qui couple une caméra classique avec une caméra infrarouge pour calculer la profondeur. Ce type de caméra est appelée RGB-D, RGB faisant référence aux caméras classiques qui créent l'image à partir trois couleurs rouge (Red), vert (Green) et bleu (Blue) et D faisant référence à la profondeur (Depth), calculée par la caméra infrarouge.

La caméra LiDAR (Light Detection and Ranging) désigne l'ensemble des caméras fonctionnant par infrarouge, cette technologie a rapidement été écartée dans la mesure où la Kinect proposait cette même technologie. Enfin la caméra Flir est une caméra thermique, son cas n'a donc pas été étudié car son utilisation ne répondait pas aux besoins.

Pour finir, le dernier critère, et sûrement le moins important de tous était le langage de programmation utilisé par les différentes librairies. En effet, mes connaissances en programmation s'arrêtent à l'utilisation de Python et de C++ et il aurait été une perte de temps de se pencher dans des fichiers écrits en Go alors que d'autres librairies peuvent proposer les mêmes solutions dans un langage qui m'est familier. Les langages Python et C++ étant les plus courant en robotique, cela n'a pas été pénalisant pour la grande majorité des librairies, seule la librairie *Localizer* a été écarté

Langage	C++	Python	Go
OpenPose		X	
OpenNI	X	X	
OpenFace		X	
MediaPipe	X	X	
AlphaPose		X	
MoveNet		X	
Flexible Collision	X		
OpenCV	X	X	
MocapNET	X		
Localizer			X
MonoCon		X	
RoboFlow			
TensorFlow		X	
VIM	X		

FIGURE 4 : Langage de programmation utilisé dans les librairies

3.3 Résultats

A partir de la comparaison entre les différentes librairies, il a pu être établi deux besoins essentiels et assez faciles à mettre en place au vu des librairies qui le proposent : la détection de visage et de pose et la détection d'objet. En effet, ces solutions sont présentes de nombreuses librairies et celles-ci présentes de nombreux avantages d'installation, de précision,... Enfin, ces deux technologies semblent pouvoir répondre à deux nombreux besoins, notamment dans le cas des travaux de détection de legos et du regard.

4 Detection de posture et de visage

4.1 librairies sélectionnées

La détection de posture et de visage est assurée par de nombreuses librairies comme l'a démontré l'état de l'art. Néanmoins certaines librairie qui paraissait bien correspondre sur le papier se sont révélées inadaptées lors de l'utilisation.

C'est le cas notamment avec OpenPose, cette librairie a une incompatibilité avec la librairie Anaconda, cependant, cette dernière m'était indispensable pour installer les librairies (voir partie 5.2 pour plus de renseignement sur l'utilisation d'Anaconda). Ainsi, il m'était impossible de l'installer et elle a aussitôt été écartée. Après des essais d'installation des différentes librairies qui semblaient performantes, MediaPipe et AlphaPose étaient les deux plus prometteuses de par leur précision, la rapidité d'exécution et la facilité d'accès aux programmes. Il fallait donc mettre en place un protocole afin de les comparer, notamment sur la précision de chacune. En effet, grâce aux codes d'exemples, la rapidité d'exécution était sensiblement la même et la facilité d'installation était également similaire.

4.2 Comparaison AlphaPose /Mediapipe

Afin de mener au mieux la comparaison entre les deux librairies Alphapose et Mediapipe, il a fallu découper cela en plusieurs travaux. En effet, la détection de main, de squelette ou de visage n'utilisent pas les mêmes algorithmes et pas les mêmes modèles de points de référence (ou *keypoints*).

Pour étayer mon propos, il faut avant tout expliquer de quelle manière fonctionnent ces algorithmes, ceux-ci utilisent des modèles qui construisent le corps, ou la partie du corps, à partir de plusieurs points de références disposés de manière intelligente (sur les articulations et points facilement identifiables comme les yeux). L'algorithme va ensuite pouvoir comparer ce qu'il a à l'image avec les modèles et définir si les points de références sont observés par l'algorithme et avec quel indice de confiance. La figure 5 montre les différentes configurations possibles par la librairie AlphaPose, ceux-ci s'appuient directement sur une autre librairie, *Halpe*. Alphapose propose aussi de travailler avec d'autres algorithmes de configuration tels que *Coco* ou d'autres mais ceux-ci n'ont pas été exploré par manque de temps.

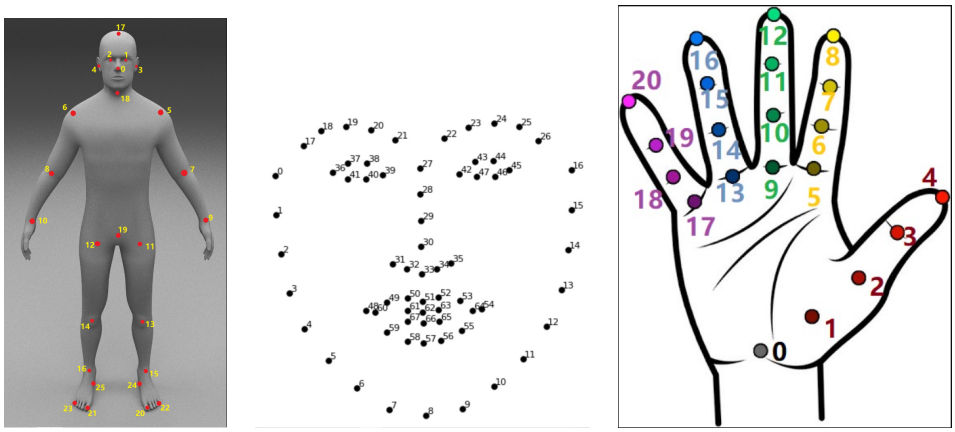

FIGURE 5 : Différents modèles de keypoints proposé par Halpe et AlphaPose

On remarque donc que chaque modèle va nous être utile selon que l'on veut détecter le visage, les mains ou le corps tout entier. AlphaPose propose également de détecter les trois zones à la fois, utilisant les trois modèles dans le même programme.

4.2.1 Detection de main

La détection de main a été la plus aisée et la plus rapide, je vais vous y expliquer la méthodologie. Les parties des programmes réalisés par moi-même sont à retrouver en annexe.

L'idée était de comparer les points relevés par l'algorithme et de les comparer à la réalité. Afin d'éviter de faire le calcul pour tout les keypoints, j'ai supposé que la différence entre les points et la réalité était globalement la même quelque soit le keypoint choisi. J'ai donc, par souci de facilité, choisi le haut de l'index gauche.

L'idée est assez simple, je crée une vidéo où avec mon index gauche, je suis une figure tracée sur mon écran. Ensuite, l'algorithme analyse la vidéo et je récupère les coordonnées pour calculer la distance avec la figure. J'ai donc fait cette méthode avec trois figures différentes, une ligne, un cercle et un carré. Trois essais étaient réalisés pour chaque figure afin d'obtenir un résultat plus fiable.

La figure 6 montre le résultat après calcul de l'algorithme. En bleu est la figure à suivre et les points de couleurs, allant du jaune au vert, sont les points calculés par l'algorithme (ici la librairie MediaPipe). Plus les points sont verts, plus la confiance donnée par l'algorithme sur les coordonnées du point est haute.

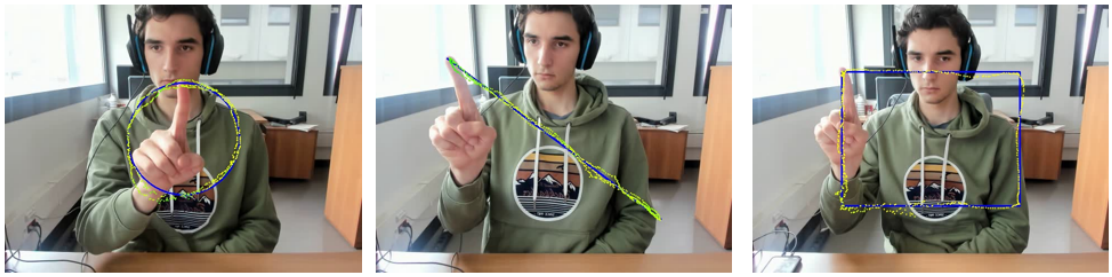

FIGURE 6 : Figure à suivre (en bleu) et points calculés par la librairie MediaPipe (en teinte de vert et jaune)

Les résultats ont ensuite été compilé dans un tableur (figure 7), où la différence moyenne de chaque figure a été calculé ainsi que la moyenne globale, afin de déterminer quelle librairie était la plus précise. Les tableurs montrant tous les résultats sont également à retrouver en annexe.

La précision est ici plus mise en avant que les autres critères de performances car ceux-ci se valent entre les deux librairies et sont a minima satisfaisants.

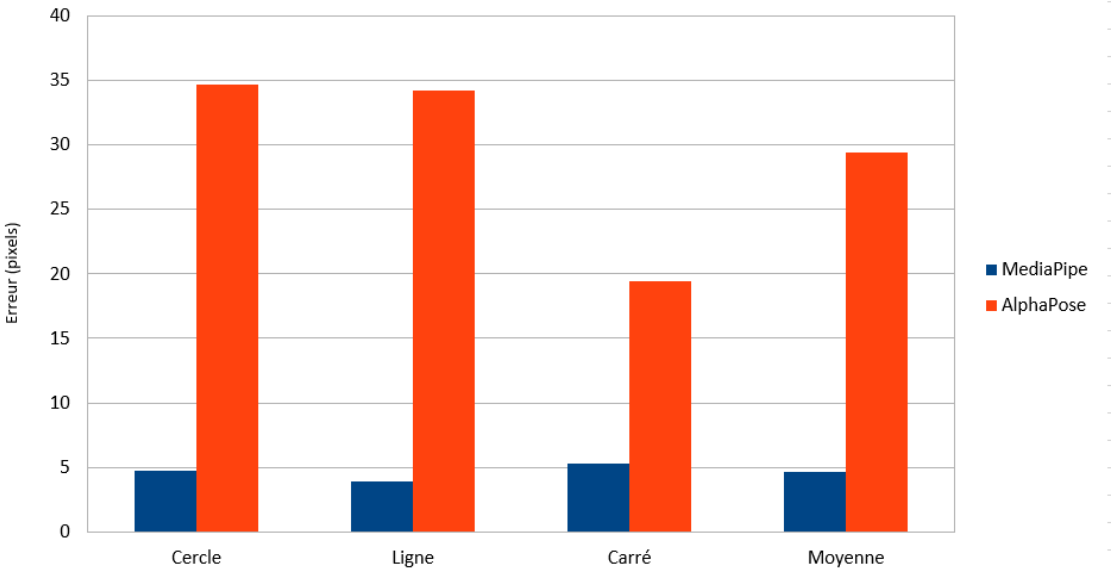

FIGURE 7 : Comparaison de performance entre les librairies AlphaPose et MediaPipe pour la détection de main

On remarque donc que MediaPipe est bien plus performante que AlphaPose pour la détection de main. Cela s'explique car AlphaPose a tendance à confondre le bout de l'index avec la jointure du pouce quand la main n'est pas correctement présentée face à la caméra. L'erreur sur le graphe s'exprime en pixel, sachant que les images sélectionnées sont dans un format standard, c'est-à-dire 640x480 pixels. MediaPipe a donc une erreur moyenne en dessous de cinq pixels, ce qui est très satisfaisant.

De plus l'erreur relevée est en partie due aux imprécisions de la méthodologie, en effet même si il est assez aisé de suivre un ligne affichée sur un écran avec le doigt, il y a quand même de faible écarts. Ces écarts s'agrandissent lorsque la figure se complexifie, comme une cercle ou un carré. C'est ce qui explique pourquoi l'erreur est plus faible pour la librairie MediaPipe lorsqu'il s'agit de suivre une ligne.

Pour AlphaPose, ces imprécisions de quelques pixels lors du relevé de points réels sont insignifiantes par rapport aux écarts de l'algorithme.

4.2.2 Détection de squelette

La comparaison des deux librairies pour la détection de squelette emploie globalement la même méthode que pour la détection de main, à la différence qu'il n'est cette fois pas possible de faire suivre à une articulation une forme donnée. Ainsi, j'ai sélectionné une vidéo sur internet d'un danseur, afin d'avoir des positions peu courantes et tester les limites des algorithmes. J'ai ensuite, image par image, relevé les coordonnées de huit points sur le corps, les coudes, les épaules, les hanches et les genoux (figure 8). Les algorithmes ont ensuite effectué leur détection et j'ai donc comparé ces points avec ceux que j'avais relevé (figure 9).

FIGURE 8 : Images avec keypoints calculés par la librairie MediaPipe

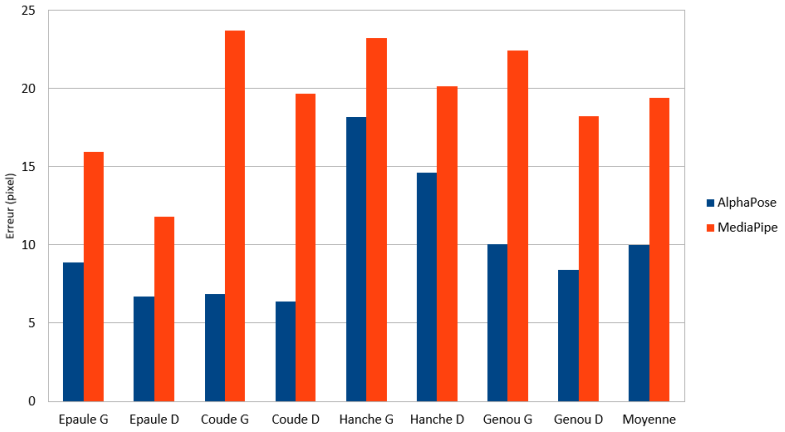

FIGURE 9 : Comparaison de performance entre les librairies AlphaPose et MediaPipe pour la détection de pose

Pour la détection de pose, on remarque que la librairie AlphaPose est bien plus performante, cela s'explique car la détection de chaque point est indépendante des autres. L'algorithme cherche à détecter séparément le genou droit, le genou gauche, etc tandis que la librairie MediaPipe va détecter toutes les articulations en même temps, si bien que si certaines d'entre-elles sont cachées ou difficilement détectables, elles seront quand même affichées, du moment que l'indice de confiance pour la détection de la pose est assez élevé. Cela conduit donc la librairie à afficher tous les keypoints, quitte à ce que certains soient imprécis.

Cependant, l'erreur d'AlphaPose reste assez élevée, cela peut s'expliquer pour deux raisons. Premièrement, le danseur fait des figures, ce qui rend la détection des keypoints plus difficiles et par conséquent, réduit la précision. Ensuite, AlphaPose est victime de ses performances, en regardant de plus près les coordonnées relevées par la librairie, on se rend compte que l'algorithme détecte parfois les épaules des personnes dans le public, pourtant plongé dans le noir. Bien que j'ai ajouté une distance maximale pour qu'un point soit valide, cela n'a pas tout supprimé. Ainsi, cela a augmenté l'erreur moyenne d'AlphaPipe mais pas assez significativement pour la rendre moins performante que MediaPipe.

4.2.3 Détection de visage

La méthodologie de comparaison de précision reste la même pour la détection de visage. La vidéo utilisée a été faite sur moi-même (figure 10), utilisant cette fois cinq keypoints, le nez, les yeux et les oreilles (figure 11).

FIGURE 10 : Images avec les keypoints calculés par la librairie AlphaPose

On remarque que la librairie AlphaPose est bien plus performante pour les mêmes raisons que pour la détection de pose. MediaPipe force à afficher tous les keypoints, impactant la précision. De plus, même lorsque que le visage est correctement présenté à la caméra, bien en face, la librairie MediaPipe n'est pas aussi précise que AlphaPose, son algorithme est donc moins performant même en situation idéale. Cet écart de précision s'accentue lorsque le visage est sur le côté ou pire, tourné vers le ciel. Les yeux et les oreilles ne sont plus visibles ou seulement partiellement et les erreurs grossières se multiplient pour MediaPipe.

4.3 Conclusion

Après la comparaison des deux librairies, il a été décidé de les utiliser toutes les deux, AlphaPose pour la détection de visage et de pose et MediaPipe pour la détection de main.

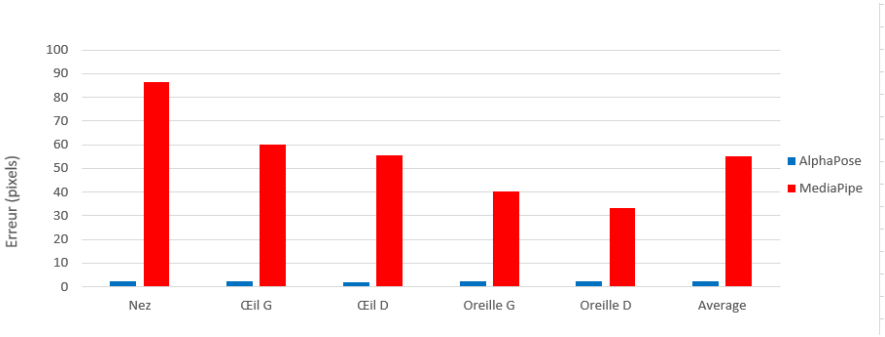

FIGURE 11 : Comparaison de performance entre les librairies AlphaPose et MediaPipe pour la détection de visage

Bien que cela exige deux environnements différents (voir partie 5) pour faire fonctionner les algorithmes et une plus grande complexité à les lancer simultanément. Néanmoins, les écarts de performances sont trop importants pour se permettre de n'utiliser qu'une seule librairie.

5 Detection d'objets

La deuxième partie de mon stage s'est axé autour de la détection d'objets, en particulier la détection de lego de différentes couleurs : bleu, rouge, vert et jaune. Cela s'inscrit dans les travaux de Benjamin CAMBLOR, comme expliqué auparavant. Durant l'expérience, le cobot doit être capable d'identifier les legos afin de pouvoir les attraper.

5.1 Librairies sélectionnées

Les librairies proposant de la détection d'objet sont moins accessibles que celles proposant de la détection humaine. Le choix est donc plus restreint et cela demande plus de travail sur la librairie en question pour arriver à une solution répondant aux besoins du laboratoire. La reconnaissance d'objet se fait sur certains objets, déjà déterminés. Cela s'explique car le processus d'apprentissage est assez long et est difficilement adaptable d'un objet à l'autre, nous y reviendrons plus tard. Ainsi, l'écrasante majorité de ces librairies proposent des détections de voiture, de piétons et de cycliste, répondant au besoin le plus demandé. Ces technologies se retrouvent principalement dans les voitures autonomes, en plein essor. Ainsi, la librairie MediaPipe, bien qu'elle propose plus de 80 objets à détecter, n'est pas adaptée. Ses performances en détection d'objets sont très satisfaisantes mais il semble dur de pouvoir entraîner l'algorithme sur un nouvel objet et les aides sur internet sont partielles et rares.

Les deux librairies retenues ont donc été TensorFlow, librairie sur laquelle s'appuie entre autres MediaPipe, qui est librairie en open-source conçue pour du machine learning. Bien plus simple à prendre en main et avec une plus grande présence de tutoriels complets sur internet, elle est bien mieux adaptée au problème.

La librairie VISp a aussi été retenue car elle permet de détecter des objets en 2D et en 3D sans utiliser de machine learning, cela évite donc une longue phase d'apprentissage.

Malheureusement, les performances en 2D ne sont pas suffisantes pour pouvoir les utiliser et la mise en place de caméra 3D telle que la Kinect est assez complexe. J'ai donc préféré rester sur TensorFlow qui s'est révélé par la suite suffisant.

5.2 Concept de Machine Learning

La détection d'objet avec TensorFlow nécessite d'utiliser du machine learning, un concept présent dans l'intelligence artificielle. Cela permet à un algorithme d'apprendre de nouvelles données sans pour autant réécrire le programme, mais en lui fournissant des données pour qu'il puisse apprendre. Cela se traduit en détection d'objets par une banque d'images, représentant l'objet à détecter. Ainsi, l'algorithme va pouvoir s'entraîner et apprendre à détecter un objet sur lequel il n'était pas forcément conçu au départ. Cela permet une grande flexibilité dans l'apprentissage, l'algorithme aurait pu aussi bien apprendre un lego, comme cela a été le cas, que n'importe quel autre objet. La difficulté réside alors autre part : dans la création de la banque de donnée. Celle-ci doit être la plus juste possible afin d'éviter plusieurs biais d'apprentissage et de réduire au maximum le temps d'apprentissage en garantissant les mêmes performances. La seconde difficulté est de déterminer les paramètres d'apprentissage correctement. Ces deux points vont être détaillés ci-dessous.

5.2.1 Création d'une banque de donnée

La création d'une banque de donnée est crucial dans le machine learning. Cela impacte directement les performances de l'algorithme et je l'ai appris à mes dépends. En effet, il n'y a pas de méthode universelle pour créer une banque de donnée, cela dépend de l'utilisation de l'algorithme et du contenu de la banque de donnée. Il y a en revanche une règle qui semble se répéter, une banque de donnée n'est efficace qu'à partir d'un millier de données. Il est difficile de faire un programme performant avec une banque plus petite. Dans mon cas, je devais donc constituer une banque d'un millier d'images, ce qui est assez long. J'ai donc fait le choix de prendre une vidéo que je découpe ensuite en images, à la fréquence de 25 images par secondes. Cela me permet également d'avoir le lego sous tous les angles, ce qui est important pour que l'algorithme puisse le détecter, quelque soit sa position et son orientation.

N'ayant aucune connaissance en machine learning au début du stage, j'ai choisi d'entraîner l'algorithme avec seulement des legos bleus. Pour que les images soient lisibles par l'algorithme, il est important de préciser où se trouve le lego, en l'enfermant dans une boite dont on renseigne les coordonnées. Ainsi, la première version de la banque de donnée était entièrement composée de lego bleu dans le même environnement, puisqu'elles étaient toutes tirées de la même série de vidéos.

Cela amène un problème assez facilement prédictible, l'algorithme est précis mais très peu adaptable. Dès que le lego sort de l'environnement dans lequel il a été pris en photo, la taux de confiance de détection chute. De plus, l'algorithme ne détecte en réalité pas que les legos bleus, il détecte également tout les objets s'approchant globalement d'un pavé droit et de couleur bleu. En effet, l'erreur dans cette banque de donnée a été de ne pas présenter d'objets bleus qui n'étaient pas des legos.

Pour construire une banque de donnée fiable, il est important de dire ce que l'algorithme doit détecter mais également ce qu'il ne doit pas détecter. Ainsi, en plaçant des objets ressemblants à des legos, la précision de l'algorithme augmente mais il faut néanmoins faire attention à ce que l'algorithme ne devienne pas trop sélectif. Il faut lui laisser une marge pour des situations sur lesquelles il n'est pas entraîné et qu'il puisse néanmoins arriver à détecter le lego. Si la définition d'un lego est trop précise et ne lui laisse pas assez de marge, il ne sera pas du tout adaptable à d'autres situations.

Ainsi j'ai amélioré la banque d'images en ajoutant des objets bleus, arrivant à un résultat satisfaisant pour les legos bleus (figure 12).

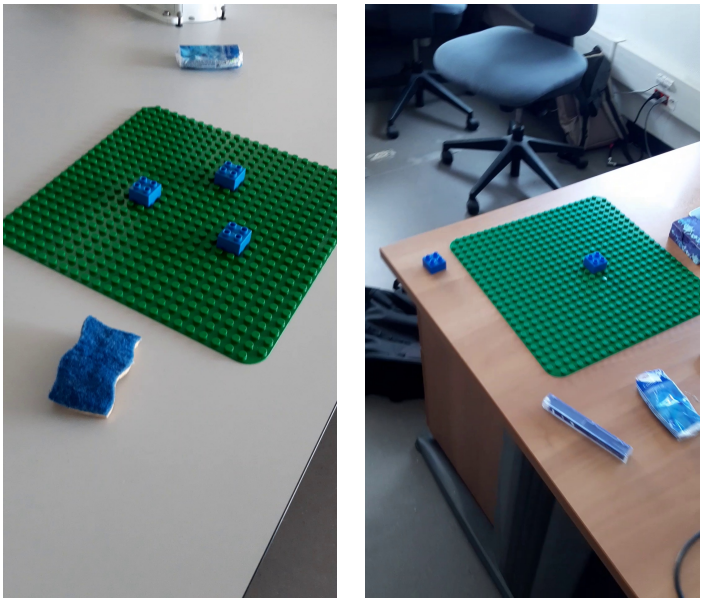

FIGURE 12 : Exemple d'images de legos bleus de la base de donnée

J'ai donc essayé ensuite d'ajouter des photos de legos rouges dans la banque de donnée, en suivant la même technique que pour les legos bleus. Ainsi, l'algorithme allait pouvoir détecter deux legos de couleurs différentes, malheureusement, la banque de donnée a été mal construite. Elle était constituée de photos de legos bleus et des photos de legos rouges, comme dit plus haut mais aucune photo incluant les deux couleurs de legos. Ainsi, l'algorithme était parfaitement capable de détecter un lego bleu, ou rouge, isolé mais dès que les deux legos de couleurs apparaissaient à l'image, le score de confiance baissait drastiquement.

Il a fallu ainsi compléter la banque d'image avec des photos où plusieurs legos de couleurs étaient présents (figure 13).

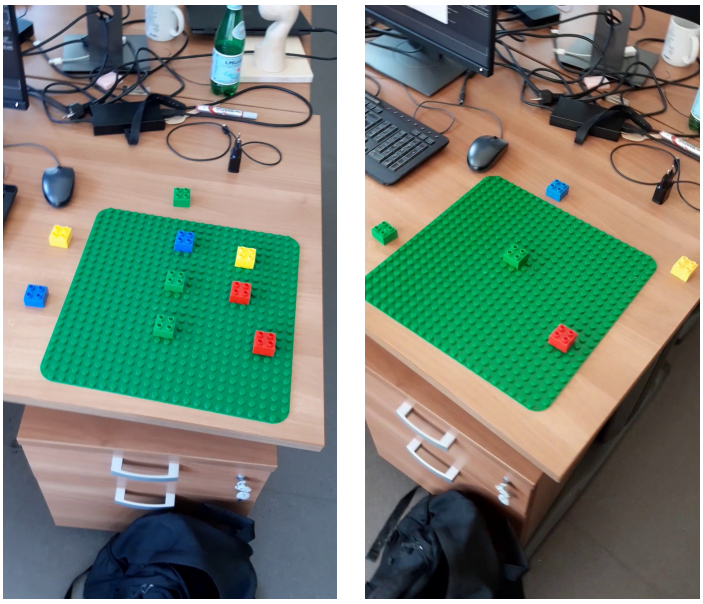

FIGURE 13 : Exemple d'images de legos de la base de donnée

On remarque sur les images qu'une grande plaque de lego verte est utilisée comme support. C'est le support qui sera utilisé pendant les expériences, ainsi, il faut entrainer l'algorithme à différencier la plaque des legos, d'autant plus avec les legos verts qui peuvent être facilement confondus. Cela n'a pas posé de problème pendant l'apprentissage mais une solution possible aurait été de peindre la plaque en noir afin de plus facilement détecter les legos posés dessus. Cette solution n'a pas été choisi car l'algorithme arrivait à détecter les legos verts sur la plaque et la banque d'image était bien avancée, la recommencer aurait pris trop de temps.

Sur toutes les images présentées, les legos sont à plats, facilement identifiables. IL y a néanmoins une configuration que l'algorithme doit apprendre qui est bien moins évidente, les legos empilés. Selon l'angle de vue, les legos ne sont pas visibles, la forme détectable est variable. Il n'est dans un premier temps plus possible de labelliser les legos par un carré. Cela fonctionnait tant qu'aucun legos n'étaient pas collés ou empilés.

Grâce à un autre logiciel de labellisation, nous avons pu encadrer les legos par des polygones, s'approchant au mieux de leur forme. Cela demande plus de temps mais permet une plus grande précision (figure 14).

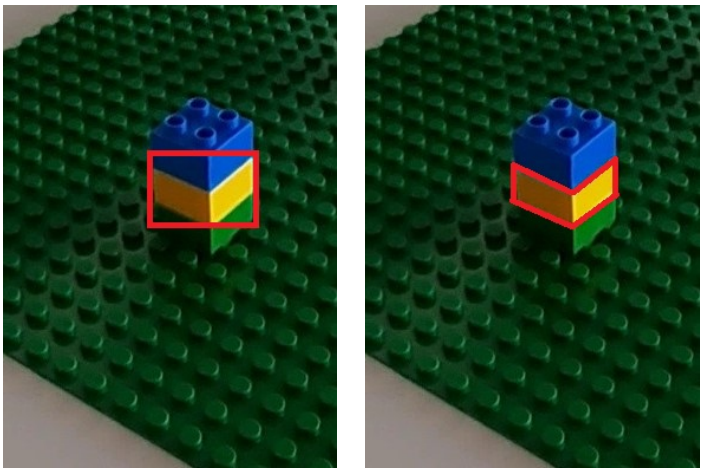

FIGURE 14 : Ancienne et nouvelle labelisation pour les legos

On voit rapidement le problème de l'ancienne labellisation sur les legos empilés. Cela empiète sur les autres legos, et le lego bleu et vert occupe la moitié du label du lego jaune, menant à des erreurs lors de la détection.

La labellisation avec un polygone évite cela et permet une détection des legos empilés bien plus précise. Les seuls problèmes observés sont quand l'angle de vue est trop à la verticale, ainsi les legos à la base de la pile sont à peine visible, rendant leur détection impossible par l'algorithme.

Une fois ces images ajoutées à la banque de données, il me semblait que l'algorithme était complet, il détectait les legos, empilés ou non, avec une précision très satisfaisante. C'est en pointant par hasard la caméra sur mon visage que j'ai découvert un autre biais de l'algorithme, ce dernier détectait mon visage comme un lego bleu. En essayant avec d'autres objets ou même des plans large de la salle, je me suis rendu compte que l'algorithme cherchait à détecter des legos même s'il n'y en avait pas.

La solution a donc été de prendre des photos sans legos et d'indiquer à l'algorithme qu'il n'y avait aucun lego à détecter sur l'image. En effet, l'intégralité des images comportait des legos, ce qui a semé la confusion dans l'algorithme, il avait ainsi appris qu'il y avait toujours au moins un lego à détecter.

Il a fallu ajouter un bon nombre d'images sans lego dans la banque de données, en effet le nombre d'images que l'on ajoute dans la banque a son importance. Plus une caractéristique est présente dans une image et récurrente de manière générale dans la banque d'images, plus elle va avoir de poids pour l'algorithme. Ainsi, ajouter beaucoup d'images sans lego permet d'apprendre à l'algorithme que des situations sans legos existent et ne sont pas rares.

C'est avec tout ces ajouts progressifs que je suis enfin arrivé à une banque d'images complète et performante, celle-ci est composée de plus de 2,000 images, ce qui, dans le milieu de la reconnaissance d'objets est assez faible. Les algorithmes de détection utilisés dans les voitures autonomes sont entraînés avec plus de 200,000 images, mais l'erreur acceptée est bien plus faible voire inexistante. De plus, la détection de lego est bien plus

simple, je l'ai réduite à une forme bien précise et à seulement quatre couleurs, ce qui n'est pas possible dans le cas de voitures ou de piétons.

5.2.2 Définir les hyperparamètres

Avoir une bonne base de donnée n'est pas suffisant pour que la détection se fasse de manière précise. En effet, la phase de l'apprentissage repose sur beaucoup de paramètres, dont les hyperparamètres, présents dans le code pour définir l'apprentissage de l'algorithme.

L'un des premiers paramètres à configurer est la répartition au sein de l'apprentissage. En effet, pour un apprentissage optimal, les données sont reparties dans trois catégories, *train*, *validation* et *test*.

La catégorie *train* représente les images sur lesquelles l'algorithme va s'entrainer et apprendre. La catégorie *validate* va permettre, au cours de l'apprentissage d'affiner les hyperparamètres afin que l'apprentissage soit plus performant. L'algorithme va s'exécuter sur la catégorie *validation*, et selon les résultats, s'affiner pour augmenter sa précision. Cela est généralement fait plusieurs fois au cours de l'apprentissage. Enfin, la catégorie *test* n'est utilisée qu'une seule fois en fin d'apprentissage pour déterminer des performances de l'algorithme. Il est important que la catégorie *validation* et *test* ne soient pas confondues pour éviter des biais d'apprentissage. En effet, si l'algorithme s'affine sur une banque d'images, il est important de disposer d'une autre banque pour vérifier ses performances.

Ainsi, il est assez évident que la catégorie *train* doit être la plus importante, on rappelle que l'algorithme doit disposer d'une importante quantité de donnée pour être performant. Les catégories *validation* et *test* peuvent être plus petites, il s'agit de vérification, il est inutile de les surcharger. Il faut cependant s'assurer que toutes les configurations y sont bien présentes.

Dans mon cas, j'ai décidé d'allouer 70% dans la catégorie *train* et 15% dans les catégories *validation* et *test*, tel que c'est fait dans la majorité des algorithmes de ce genre.

Ensuite, un autre paramètre très important est le nombre d'époques que va faire l'algorithme, c'est-à-dire le nombre de fois, où, après s'être entraîné sur la catégorie *train*, l'algorithme va se tester sur la catégorie *validation* et donc affiner ses hyperparamètres. Il faut faire attention à ce que cela ne soit pas trop faible, ou l'algorithme ne sera pas assez précis et ni trop élevé, auquel cas, l'algorithme sera sur-entraîné, et il aura plus de mal à reconnaitre les legos dans des configurations différentes de celles qu'il a déjà rencontré. De plus, le temps de calcul se trouve rallongé.

Après plusieurs essais à 20000 époques, j'ai décidé de le réduire à 1000 époques, ce qui rendait l'apprentissage bien moins long, et l'algorithme plus performant. Il est difficile de descendre en dessous, l'algorithme a alors du mal à bien détecter les legos, cela est sans doute du à la banque d'images à peine assez grande.

Les hyperparamètres peuvent être également réglés, mais ceux-ci ont un impact plus limité car ils sont modifié pendant l'apprentissage. Il y a, entre autre, le *learning rate* (taux d'apprentissage), qui détermine à quel point les erreurs constatées pendant l'apprentissage vont pouvoir changer les hyperparamètres. Encore une fois, il faut trouver

le juste milieu, un *learning rate* trop faible peut mener à un algorithme trop rigide, qui demande un temps d'apprentissage bien trop long pour arriver à un résultat satisfaisant. A contrario, un *learning rate* trop élevé peut rendre l'algorithme trop instable.

5.3 Conclusion

La librairie TensorFlow est assez flexible et performante pour en faire un outil efficace et utilisable dans ce contexte. C'est pour cela qu'elle a été choisi à défaut des autres librairies. En effet, avec le bon paramétrage et une banque de données bien construite, les résultats sont satisfaisants (figures 15, 16).

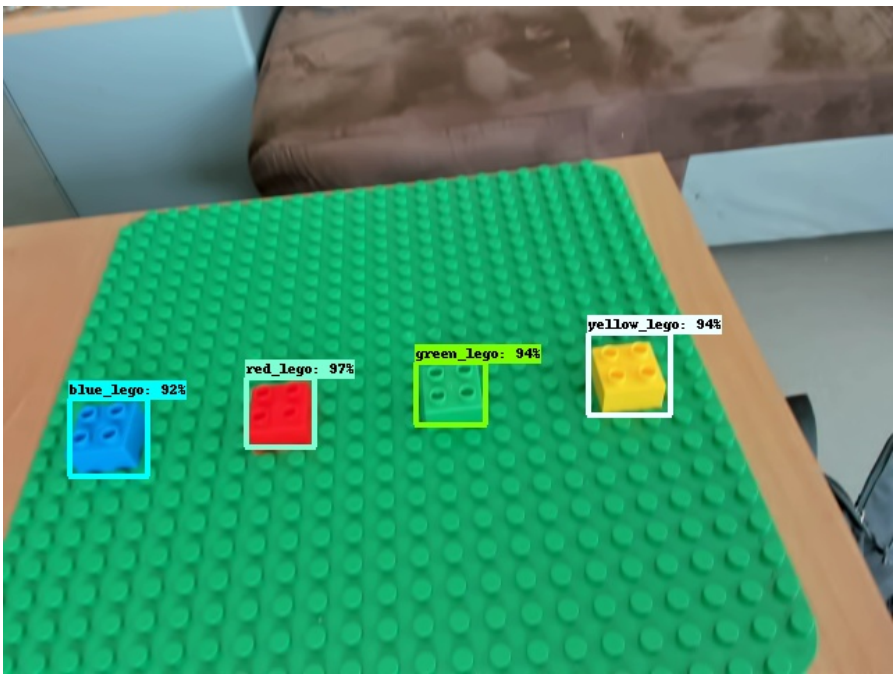

FIGURE 15 : Détection de legos avec TensorFlow

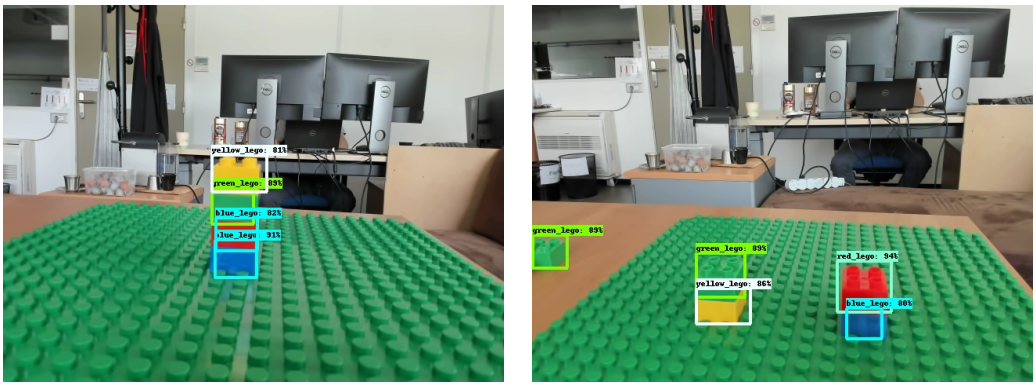

FIGURE 16 : Détection de legos empilés avec TensorFlow

6 Transmission des librairies

Une fois les librairies sélectionnées et les algorithmes retravaillés pour répondre aux besoins spécifiques du laboratoire, il est important de pouvoir transmettre ce travail de la manière la plus simple possible. N'importe qui doit pouvoir être capable de réutiliser ce que j'ai fait, à n'importe quel moment, en garantissant une simplicité et une rapidité d'utilisation.

6.1 GitLab

La solution la plus couramment utilisée est le site GitLab (ou GitHub). Ces deux sites fonctionnent de la même manière et reposent sur le même principe, un espace de stockage qui sert à partager des librairies informatiques, appelé généralement un *git*. C'est notamment grâce à ce site que j'ai pu installer et travailler sur des librairies comme TensorFlow, AlphaPose, MediaPipe,...

Ainsi, le dossier GitLab s'accompagne toujours d'un fichier *README.md*, un fichier qui permet de donner les instructions pour installer de manière simple et rapide la librairie.

Néanmoins, l'installation d'un git peut s'accompagner de difficultés variées, généralement due à des incompatibilités entre la configuration de la machine de travail et la configuration requise pour la librairie.

Il semble peu raisonnable de devoir reconfigurer son ordinateur à chaque installation de librairie, d'autant plus quand certaines librairies demandent des configurations incompatibles entre elles. L'une des solutions est alors de passer par les environnements virtuels.

6.2 Docker

La création d'environnements virtuels est assez courant lors de l'installation de librairie via des git. Cela est souvent préconisé pour simplifier l'installation.

Malheureusement, les environnements virtuels ne suffisent pas toujours, en effet un environnement virtuel est une configuration annexe, qui ne dépend pas de celle utilisée sur la machine. Les installations de librairies qui y sont faites n'impactent pas la configuration de base de l'ordinateur. Cela est donc utilisée pour y installer des versions bien précises, parfois antérieures à celles déjà installées sur l'ordinateur. Il est possible de créer autant d'environnement virtuels que possible et d'en lancer plusieurs à la fois. On peut donc créer un environnement virtuel pour chaque librairie que l'on veut installer.

Cet outil avait été largement utilisé durant mon début de stage, notamment avec la librairie Anaconda qui permet de créer rapidement et facilement des environnements virtuels avec le langage informatique Python déjà installé.

Cependant, les environnements virtuels ne sont pas totalement hermétiques à la configuration de l'ordinateur. Ils en conservent certaines propriétés qui peuvent s'avérer bloquantes pour l'installation de librairies, comme ça a été le cas pour les librairies utilisées pendant ce stage.

Il est donc nécessaire d'employer un autre type d'environnement virtuel, le *docker*. Le docker est une technologie qui repose sur deux concepts, un *container* et une *image*.

L'image est une configuration donnée, qui peut être utilisée dans un container. Ainsi, le container est la boite et l'image est ce que l'on va mettre dans la boite. Cette boite est totalement hermétique du reste de l'ordinateur, un nouvel espace de travail est créé avec une image pouvant être totalement différente de la configuration de l'ordinateur.

Cela demande alors un plus grand travail de reconfigurer entièrement l'image pour pouvoir l'utiliser avec les librairies, il existe cependant des images toutes faites qui fonctionnent avec un librairie donnée. C'est donc avec cela que j'ai pu créer des images qui collaient parfaitement à l'utilisation des librairies et qui avaient la configuration nécessaire.

La création d'images se fait relativement facilement, et cela assure que, quelque soit la machine utilisée, la librairie sera opérationnelle. Je n'ai donc eu aucun soucis à créer les images pour les différentes librairies hormis AlphaPose, en effet, la configuration de carte graphique est assez complexe lorsqu'on se trouve à l'intérieur d'un container et c'est cela qui m'a posé problème. Ces problèmes sont arrivés durant ma dernière semaine de stage, je n'ai malheureusement pas eu le temps de les régler, j'ai donc du revenir à une solution moins sure, les environnements virtuels. Ceux-la même sont proposés sur le git d'AlphaPose mais j'ai rencontré nombres de problèmes lorsque j'ai installé cette librairie et il n'est pas exclu que ces complications se répètent sur d'autres machines.

7 Conclusion

7.1 Résultats au regard des objectifs

En conclusion de ce rapport, les objectifs ont été atteint dans l'ensemble dans la mesure où ceux-ci étaient assez flous. Je n'avais pas de limite précise à mon stage, seulement apporter de nouvelles solutions dans la vision, la perception et la détection pour l'utilisation de cobot. J'ai eu des demandes plus précises, comme pour la détection de legos mais bien évidemment ce n'est qu'un petit apport sur l'ensemble de ce que peuvent fournir ces librairies en terme de vision.

Par exemple, une approche sur une détection 3D permettrait sûrement de réduire les erreurs de détection et d'estimation de positionnement dans l'espace. Il aurait aussi pu être intéressant de travailler avec d'autres modèles de machine learning pour pouvoir comparer leur performance.

Ainsi, les objectifs fixés au fur et à mesure du stage ont été réalisé dans l'ensemble, seule la librairie AlphaPose n'a pas pu être transmise comme expliqué plus haut.

7.2 Apports du stage

Ce stage m'a beaucoup apporté dans des domaines techniques, notamment sur le machine learning et les technologies autour de la vision, deux domaines qui m'étaient assez

étrangers. J'ai également découvert des outils comme les dockers ou plus généralement les environnements virtuels, qui, au vu de leur puissance, me seront utiles par la suite.

Ce stage en laboratoire m'a également permis de découvrir le monde de la recherche qui m'était jusqu'alors assez inconnu. Je n'avais jamais travaillé dans ce milieu et j'y ai donc découvert des problématiques qui lui y sont propres.

J'ai enfin approfondi mon autonomie, avec une méthode de travail qui m'était relativement nouvelle. Jusqu'alors, mes travaux dans ma scolarité restaient assez guidés, avec un cadre bien défini. Cela était bien moins le cas durant mon stage. Bien sur, il y avait un cadre dans lequel je me devais de rester, mais n'ayant pas le temps de tout réaliser, il m'a fallu faire des choix et s'assurer seul que c'était les bons.

Références

Site internet des git des librairies
AlphaPose : https://github.com/MVIG-SJTU/AlphaPose
MediaPipe : https://google.github.io/mediapipe/
TensorFlow : https://github.com/tensorflow/tensorflow

Annexe

Code

Programme d'obtention des coordonnées de keypoint pour la détection de doigt pour un suivi de carré

```
import cv2
import time

cap = cv2.VideoCapture(0)

width = int(cap.get(3))
height = int(cap.get(4))
frame_size = (width,height)
fps = 20

output = cv2.VideoWriter('capture_square_3.avi', ...
    cv2.VideoWriter_fourcc('M','J','P','G'), 20, frame_size)

ratio = 4

p1 = (int(width/ratio), int(height/ratio))
p2 = (int((ratio-1)*width/ratio), int(height/ratio))
p3 = (int((ratio -1)*width/ratio), int((ratio -1)*height/ratio))
p4 = (int(width/ratio), int((ratio -1)*height/ratio))

print("appuyez sur 'q' lorsque votre doigt est positionn  sur le ...
    carr . Une seule main doit apparaitre    l' cran ")
while True:
    _, image = cap.read()

    image = cv2.flip(image, 1)

    height, width, _ = image.shape

    cv2.line(image, p1, p2, (0, 255, 0), 2)
    cv2.line(image, p2, p3, (0, 255, 0), 2)
    cv2.line(image, p3, p4, (0, 255, 0), 2)
    cv2.line(image, p4, p1, (0, 255, 0), 2)

    cv2.imshow("image", image)

    if cv2.waitKey(25) & 0xFF == ord('q'):
        t0 = time.time()
        dt = 0

        print("suivez le carr  avec votre doigt")
        while True:

            _, image = cap.read()

            image = cv2.flip(image, 1)

            output.write(image)
```

```
47
48              height, width, _ = image.shape
49
50              cv2.line(image, p1, p2, (0, 255, 0), 2)
51              cv2.line(image, p2, p3, (0, 255, 0), 2)
52              cv2.line(image, p3, p4, (0, 255, 0), 2)
53              cv2.line(image, p4, p1, (0, 255, 0), 2)
54              cv2.imshow("image", image)
55
56              t1 = time.time()
57              dt = t1-t0
58              if cv2.waitKey(25) & int(dt) > 20:
59                  break
60          break
61
62  print("Fin de l'enregistrement")
63  cap.release()
64  output.release()
65  cv2.destroyAllWindows()
```

Détection des keypoints de la main avec MediaPipe

```
1   import cv2
2   import mediapipe as mp
3
4   print(mp.__file__)
5   mp_drawing = mp.hand_detection.drawing_utils
6   mp_drawing_styles = mp.solutions.drawing_styles
7   mp_hands = mp.solutions.hands
8
9   mp_hands.Hands(static_image_mode=4)
10  print(mp.__file__)
11
12  # For static images:
13  mp_model = mp_hands.Hands(
14      static_image_mode=True,  # only static images
15      max_num_hands=2,  # max 2 hands detection
16      min_detection_confidence=0.5)  # detection confidence
17
18  # we are not using tracking confidence as static_image_mode is true.
19  data = open("data.txt", "w")
20  score = open("score.txt", "w")
21  # cap=cv2.VideoCapture("capture_square_1.avi")
22  cap = cv2.VideoCapture(0)
23
24  while (cap.isOpened()):
25      success, image = cap.read()
26      if success == True:
27          # image = cv2.flip(image, 1)
28
29          cv2.waitKey(10)
30
31          results = mp_model.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

```
32
33          print(results.multi_handedness)
34
35          image_height, image_width, c = image.shape  # get image shape
36          print("HEIGHT :", image_height)
37          print("WIDTH :", image_width)
38          # iterate on all detected hand landmarks
39          if results.multi_hand_landmarks:
40              for hand_landmarks in results.multi_hand_landmarks:
41                  # we can get points using mp_hands
42                  score.write(str(results.multi_handedness))
43                  score.write("\n")
44                  data.write(str(hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_T
                        * image_width))
45                  data.write("\n")
46                  data.write(str(hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_T
                        * image_height))
47                  data.write("\n")
48                  print(f'Ring finger tip coordinates: (',
49                        f'{hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].x
                            * image_width}, '
50                        f'{hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].y
                            * image_height})'
51                        )
52              for hand_landmarks in results.multi_hand_landmarks:
53                  mp_drawing.draw_landmarks(
54                      image,  # image to draw
55                      hand_landmarks,  # model output
56                      mp_hands.HAND_CONNECTIONS,  # hand connections
57                      mp_drawing_styles.get_default_hand_landmarks_style(),
58                      mp_drawing_styles.get_default_hand_connections_style())
59
60          cv2.imshow("l'image", image)
61          if cv2.waitKey(25) & 0xFF == ord('q'):
62              break
63      else:
64          break
65
66  data.close()
67  score.close()
68  cap.release()
69  cv2.destroyAllWindows()
```

Comparaison et affichage des résultats entre la détection par MediaPipe et les coordonnées réelles

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  scores = []
6  dist = []
7  dist_8 = []
```

```
dist_95 = []
c = 0

x = None
y = None

cap = cv2.VideoCapture("capture_square_1.avi")

width = int(cap.get(3))
height = int(cap.get(4))

ratio = 4

x1, y1 = width/ratio, height/ratio
x2, y2 = (ratio -1)*width/ratio, (ratio -1)*height/ratio

ret, image = cap.read()

p1 = np.array([x1, y1])
p2 = np.array([x2, y1])
p3 = np.array([x2, y2])
p4 = np.array([x1, y2])

c1 = (int(x1), int(y1))
c2 = (int(x2), int(y1))
c3 = (int(x2), int(y2))
c4 = (int(x1), int(y2))

cv2.line(image, c1, c2, (255, 0, 0), 2)
cv2.line(image, c2, c3, (255, 0, 0), 2)
cv2.line(image, c3, c4, (255, 0, 0), 2)
cv2.line(image, c4, c1, (255, 0, 0), 2)

# acquisition des donn es
score = open("score.txt", "r")
data = open("data.txt", "r")

for f in score.readlines():
    if len(f) > 3 and f[2] == 's':
        new_f = f[9:-2]
        scores.append(float(new_f))

a = 255/(1-min(scores))
b = 255 - a

for g in data.readlines():
    new_g = g[:-2]
    if x == None:
        x = float(new_g)
    else:
        y = float(new_g)
        p5 = np.array([x, y])
        # affichage des points avec  chelle  de couleur selon la ...
            confiancede relev
        cv2.circle(image, (int(x), int(y)), 1, (0,255-(a*(1-scores[c]) ...
            + b), (a*scores[c]+b)), -1)
```

```
62
63          # calcul de la distance du point    la ligne
64          d1 = np.abs(np.cross(p2-p1,p5-p1)/np.linalg.norm(p2-p1))
65          d2 = np.abs(np.cross(p3-p2,p5-p2)/np.linalg.norm(p3-p2))
66          d3 = np.abs(np.cross(p4-p3,p5-p3)/np.linalg.norm(p4-p3))
67          d4 = np.abs(np.cross(p1-p4,p5-p4)/np.linalg.norm(p1-p4))
68
69          d = min(min(d1, d2), min(d3, d4))
70          if scores[c] ≥ 0.98 :
71              dist_8.append(np.abs(d))
72              if scores[c] ≥ 0.995 :
73                  dist_95.append(np.abs(d))
74          dist.append(d)
75          x = None
76          c += 1
77
78  cv2.imwrite('performances/image_square_1.jpg', image)
79  cv2.imshow('image', image)
80  cv2.waitKey(0)
81  cv2.destroyAllWindows()
82
83  moy = 0
84  moy_8 = 0
85  moy_95 = 0
86  moy_pond = 0
87  moy_scores = 0
88
89  for i in range(len(scores)):
90      moy += dist[i]/len(scores)
91      moy_pond += dist[i]*scores[i]/len(scores)
92      moy_scores += scores[i]/len(scores)
93
94  for i in range(len(dist_8)):
95      moy_8 += dist_8[i]/len(dist_8)
96
97  for i in range(len(dist_95)):
98      moy_95 += dist_95[i]/len(dist_95)
99
100
101 print("Nombre d'acquisitions :", len(scores))
102 print("Acquisition avec confiance au dela de 0.98 :", len(dist_8), " ...
        et au dela de 0.995 :", len(dist_95))
103 print("Moyenne :", moy)
104 print("Moyenne p o n d r e par le score :", moy_pond)
105 print("Moyenne scores :", moy_scores)
106 print("Moyenne avec confiance au dela de 0.98 :", moy_8, " et au dela ...
        de 0.995 :", moy_95)
107
108 plt.figure()
109 plt.xlabel("score de confiance")
110 plt.ylabel("distance    la droite")
111 plt.title("Distance et score")
112 plt.plot(scores, dist, 'x')
113 plt.savefig("performances/graph_square_1.jpg")
114 plt.show()
```

Besoins/Librairi	OpenPose	OpenNI	OpenFace	MediaPipe	AlphaPose	MoveNet	Flexible Collision	OpenCV	MocapNET	Localizer	MonoCon	RoboFlow	TensorFlow	VIM
Detection Mains	X	X		X	X	X								
Detection Pose 2D	X	X		X	X	X			X					
Detection Pose 3D	X								X					
Detecion Visage	X	X	X	X					X					
Detection expression			X											
Detection Focus														
Calcul de Distance				X			X							
Detection Collision							X							
Position Objet	X			X									X	X
Orientation Objet				X				X		X	X	X	X	X
Couleur Objet								X				X		
Nature Objet				X								X	X	

Langage														
C++		X		X			X	X	X					X
Python	X	X	X	X	X	X		X			X		X	
Go										X				
Incompatibilite	Anaconda													
Dependances	CUDA, cuDNN, Cmake, protobuf	LibUSB, FreeGLUT3, JDK	Cmake, OpenBLAS, OpenCV,	Bazel 5.0.0, OpenCV, ffmpeg	CUDA, Cython, Pytorch, Torchvision	Anaconda, Pytorch, COCOAPI, Cython	Eigen, libcdd, Cmake		OpenCV	TensorFlow, sync	Anaconda, Pytorch, mmcv, mmdetection, mmsegmentation		CocoAPI, ProtoBuf, OpenCV, pandas, detection Api	

Librairies/Camer	Stereo	Flir Camera	WebCam	Video	Kinect	LIDAR
OpenPose	X	X	X	X		
OpenNi					X	
OpenFace			X	X		
MediaPipe			X	X	X	
AlphaPose			X	X		
MoveNet			X	X		
Flexible Collision			X	X		X
OpenCV			X	X		
MocapNET			X	X		
Localizer			X	X		
MonoCon			X	X		
VIM			X	X	X	

Disponible	
Potentiellement disponible	
Indisponible	

Performances														
Rapidite dexecutior	+		+	++	++	++		++			--	--	--	++
Precision	+		+	++	++	+		+	++			0	0	0
Adaptibilite	0		+	+	++			0				++	+	+
Installation	0			0	0			++	+		+	++	+	++
Accessibilite		0	++	0				+			0	--	0	+
Robustesse	++			+	+			++				++	+	
Portabilite	+	++						+					+	++

Excellente	
Bonne	
Moyenne	
Mauvaise	

Tres bien documentee	
Bien documentee	
Moyennement documentee	
Peu ou pas documentee	

RAPPORT D'EVALUATION

ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
☎ 00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* centre INRIA de l'université de Bordeaux

Adresse / *Address* 200 avenue de la vieille tour 33405 Talence

Tél / *Phone (including country and area code)* _____

Nom du superviseur / *Name of internship supervisor* David DANEY

Fonction / *Function* Chercheur, responsable scientifique de l'équipe-projet Auctus

Adresse e-mail / *E-mail address* david.daney@inria.fr

Nom du stagiaire accueilli / *Name of intern* Simon GERVAISE

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)** *Please attribute a mark from **A (excellent)** to **F (very weak).***

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ? **A** B C D E F
  Was the initial contract carried out to your satisfaction?

- Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☒ non/*no*
  Was the intern lacking skills?

  Si oui, lesquelles ? / *If so, which skills?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)*
  A B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Simon s'est parfaitement intégré et a ainsi pu bénéficier de l'intérêt de l'assistance de l'équipe ainsi que de répondre aux besoins

Version du 05/04/2019

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?

Did the intern live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

Q B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* __ Son comportement a été impeccable.

_____

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? Q B C D E F (Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? Q B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* __ Il a pris l'initiative de la méthodologie et de l'ensemble des actions a mener

_____

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? Q B C D E F *Was the intern open to listening and expressing himself/herself?*

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Simon a une très bonne capacité d' écoute et une bonne ouverture.

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : Q B C D E F
Please evaluate the technical skills of the intern: Le travail fourni a été au dela de mes espérances, et réalisé avec une très grande autonomie.

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

Would you be willing to host another intern next year? ☒ oui/*yes* ☐ non/*no*

Fait à Bordeaux , le 26/09/22
In _____, on _____

Signature Entreprise _____ Signature stagiaire *Company stamp* _____ *Intern's signature*

Merci pour votre coopération
We thank you very much for your cooperation