

# Contrôle corps-complet d'un humanoïde: Commande par computer vision

Manu Serpette<sup>1</sup>

**Abstract**—Ce document propose un framework permettant de faire effectuer des tâches automatiques au robot humanoïde en utilisant la vision pour passer des commandes à un contrôleur Corps-complet.

## I. INTRODUCTION

La téléopération de robots humanoïdes est un enjeu majeur en robotique. L'équipe Larsens du LORIA développe un contrôleur pour robots humanoïdes, en particulier sur le robot Talos de PAL robotics, avec pour objectif d'assurer une téléopération temps réel par un opérateur. Les débouchés de ce projet sont très intéressants, en particulier pour accéder à des zones dangereuses qui nécessitent l'intervention d'un humain mais qu'il serait risqué d'envoyer directement, comme une centrale nucléaire par exemple. L'intérêt d'utiliser un humanoïde est que ce dernier est bien plus précis et capable de gestes techniques similaires à un humain qu'un robot à roue par exemple, de plus étant fait comme un homme il peut mimiquer les tâches humaines et faciliter l'immersion du téléopérateur. Cependant la conception de robot humanoïde capable de se mouvoir aisément dans l'espace est très complexe, la démarche bipède est pleine de difficultés car repose sur un important contrôle de l'équilibre de tout le corps chez le bipède, naturellement un humain va déplacer chaque partie de son corps chaque fois qu'un mouvement spécifique du corps est voulu, pour conserver son équilibre. Et il faut donc mimiquer cet équilibre général chez le robot humanoïde.

Dans le but d'effectuer de la téléopération avec ce contrôleur l'équipe Larsens utilise une combinaison de capture de mouvement par motion tracking [1], mais la vision du robot n'est pas utilisée autre que pour donner un retour à l'opérateur. L'objectif premier de ce travail est donc d'utiliser la vision du robot pour automatiser des tâches de téléopération. Le premier pas vers cette objectif est de pouvoir indiquer un objet au robot au travers de la caméra et que le robot suive l'objet avec son bras. Ceci serait fort utile par exemple pour faire de la prise de pouls, pour interagir avec un panneau de contrôle ou pour valider un badge. Cependant cet objectif contient plusieurs difficultés, il faut tout d'abord trouver une méthode de tracking fiable, robuste mais surtout très générale, qui soit capable de détecter et de traquer n'importe quelle forme d'objets n'importe quel amas de pixels non connu par le programme à l'avance. Il faudra ensuite également réussir à lier ce système tracking



Fig. 1. **Talos and Icube robots.** Crédit: Équipe Larsens

au contrôleur et positionner l'objet traquer dans l'espace cartésien du contrôleur.

## II. ÉTAT DE L'ART

### A. Contrôleur corps-complet

L'équilibre bipède est difficile à réaliser pour un robot. En effet pour conserver son équilibre, à chaque mouvement d'un de ses degrés de liberté, 66 pour Talos comme son bras par exemple, le robot doit déplacer tous les autres degrés de liberté pour conserver son équilibre. Il n'est donc pas possible de calculer et déplacer chaque degré de liberté l'un après l'autre sous peine de voir le robot chuter au cours de la manœuvre. Or c'est exactement ce que fait un contrôleur classique, pour atteindre une position donnée il va commander chaque degré l'un après l'autre. Ainsi l'équipe Larsens a développé son propre contrôleur [2], un contrôleur corps complet qui à chaque pas de temps recalcule la position de tous les degrés de liberté du robot en prenant en compte l'objectif et les contraintes pour avancer vers la position cible et conserver l'équilibre. Pour ce faire, le contrôleur modélise la commande du corps entier en tant que problème d'optimisation multi-tâches.

$$(\tau^*, \ddot{q}^*) = \underset{\tau, \ddot{q}}{\operatorname{argmin}} \sum_{k=0}^{n_{\text{tasks}}} w_k \|A_k(q, \dot{q})\ddot{q} - b_k(q, \dot{q})\|^2 \quad (1)$$

$$\text{s.t. } \begin{cases} A_{ineq}(q, \dot{q})\ddot{q} \leq b_{ineq}(q, \dot{q}) \\ A_{eq}(q, \dot{q})\ddot{q} = b_{eq}(q, \dot{q}) \\ \tau = M(q)\ddot{q} + h(q, \dot{q}) - J(q)^T f \end{cases}$$

Avec  $\tau^*$  et  $\ddot{q}^*$  les accélérations et couples optimaux.  $w_k$  le poids de chaque tâche. Les tâches sont pondérées au même niveau et leurs priorité dépend de  $w_k$  [3]. Ces tâches représentent plusieurs objectifs dans un espace cartésien, un

\* Sous l'encadrement de Jean-Baptiste Mouret de l'équipe Larsen

<sup>1</sup>Mines de Nancy Inria, LORIA, F-54000 Nancy, France, manu.

optimiseur trouve les valeurs de commande qui permettent à la fois d'atteindre l'objectif et de respecter les contraintes ( Ici le Eiquadprog<sup>2</sup> QP solver and Pinocchio<sup>3</sup> [4], [5] ) . Ces entrées de commande peuvent être des vitesses, des accélérations ou des couples d'articulations.

Ainsi une tâche sur un degré de liberté est modélisée par une fonction de coût, le coût augmente si la tâche n'est pas respectée et utilise un solveur quadratique dont les paramètres sont les pitchs ( joint torques and acceleration) à appliquer à chaque moteur. La tâche se voit également attribuer un coefficient, un poids (wk) qui donne l'importance de la tâche ou contrainte dans le problème quadratique. Ainsi ce solveur permet à chaque instant de déterminer l'ensemble de solutions de commandes pour chaque moteur qui atteindra un équilibre et avancera vers la position cible.

A ces tâches s'ajoutent des contraintes, également modélisées par des fonctions de coûts. Ces contraintes permettent en particulier de créer des sphères de collisions dans l'espace du robot, sphères auxquelles ont peut attribuer un poids pour effectuer une répulsion d'un degré de liberté dans le contrôleur. Cette méthode permet en particulier d'implanter des tâches d'auto-collisions pour éviter au robot de se cogner lui-même. Quand le contrôleur cherche à placer le gripper du robot à un endroit donné, il va effectuer une résolution quadratique sur toute ses contraintes pour trouver une solution. Une contrainte de collisions est représentée par une fonction de perte, par un point du robot et des sphères répartis sur le robot. Quand le point approche d'une sphère la perte augmente proportionnellement au poids, ici on utilise des

$$\exp(-x) \quad (2)$$

et donc très rapidement la perte est très grande près des sphères, ainsi le contrôleur prohibe naturellement les positions avec dans les sphères de collisions car il y a un fort gradient de répulsion au abords des sphères. De plus La décroissance exponentielle provoque une convergence rapide vers zéro quand on est loin de du centre de la sphère, ce qui signifie que les sphères de répulsion n'influencent pas les comportements du robot quand quand les sphères de sont pas proches d'un corps. La fonctions de coût choisi spécifiquement pour les tâches de répulsion est la suivante:

$$C(q) = \exp\left(-\frac{\|x - p_0\|^p}{a}\right) \quad (3)$$

Où  $a = r_{rep} + r_{frame}$ , avec  $r_{rep}$  un rayon autour d'un point de répulsion et  $r_{frame}$  un rayon autour du cadre qui doit éviter les répulseurs.  $p_0$  est le point de répulsion en coordonnées cartésiennes.  $p$  est un exposant défini par l'utilisateur.

L'avantage de ce contrôleur est qu'il est multiplateforme, il peut être déployé sur le robot Tiago pour tester les programmes avant d'aller sur le robot Talos.

<sup>2</sup><https://github.com/stack-of-tasks/equadprog>

<sup>3</sup><https://github.com/stack-of-tasks/pinocchio>

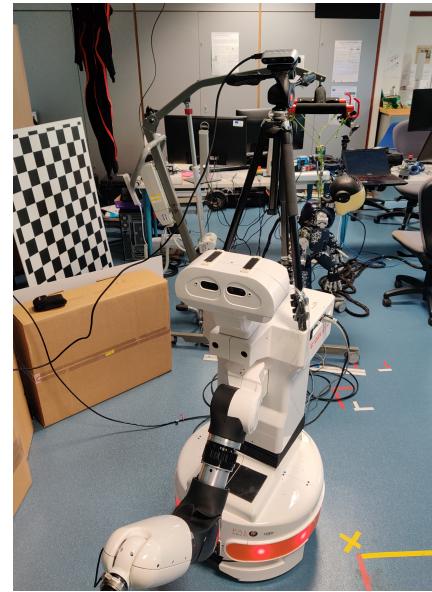


Fig. 2. Robot Tiago

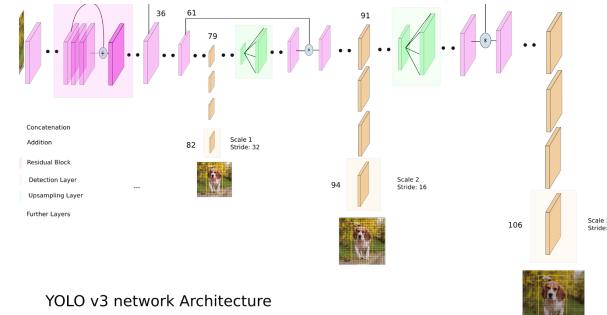


Fig. 3. Crédit: [6]

### B. Tracking

De multiples techniques existent pour le tracking comme le suivi par extraction de formes géométriques et étalonnage de la caméra ou encore l'utilisation de filtres de corrélation. Cependant la première est peu utile dans notre cas car il faut connaître à l'avance une forme précise à traquer ce qui va à l'encontre du but de ce projet, il faut ensuite faire de l'extraction de forme géométriques, l'objet ne peut quitter le champ de vision puis revenir et il faut étalonner la caméra. Et si la deuxième semble plus adaptée car les filtres de corrélation fournissent un tracking très rapide et ce sur n'importe quelle cible ils ne sont pas assez robuste, en effet ils ne supportent pas la rotations d'objet par exemple, en fait ils manquent de capacités d'adaptation, ils ne s'adaptent à l'évolution visuel de l'objet dans le temps et reste sur leur première impression..

Le besoin d'adaptation amène naturellement vers les réseaux de neurones. Les premiers algorithmes de machine learning qui semblent utiles pour du tracking sont des algorithmes de détection d'objets par convolution comme YOLO [6](cf fig 3), qui utilise une implémentation de fast R-CNN [7]. Cependant si ce réseau est performant dans sa tâche, cette dernière ne correspond pas à l'objectif fixé.

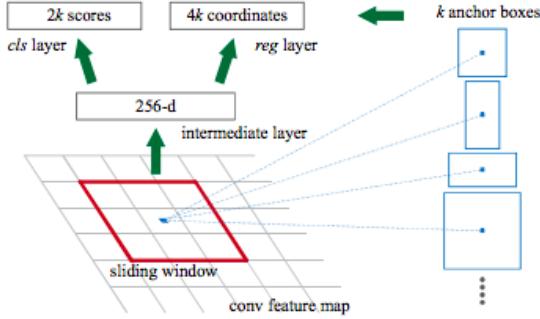


Fig. 4. Couche de RPN, Crédit: [11]

En effet YOLO permet de détecter plusieurs objets d'un coup, option superflue dans le cadre de ce projet mais surtout il ne détecte que ce pourquoi il est entraîné et ne permet de lui désigner une cible coup sur coup. Il n'est donc pas possible de remplir une fonction de tracking généralisé efficace avec ce réseau, qui plus est le "tracking" manquerait de fluidité car il sautera quand yolo ne sera pas sûr de ce qu'il détecte ce qui est problématique pour le mouvement du robot, mais l'implémentation du fast R-CNN de YOLO, qui permet d'utiliser les capacités de reconnaissance des R-CNN [8] en temps réels, reste toujours intéressante pour le problème de tracking de cette article si on peut détecter des formes quelconque .

Une idée permettant d'exploiter ces algorithmes pour le tracking est de les appliquer à la détection de pattern [9] dans une image. L'idée est de pouvoir détecter la position probable d'un pattern de pixels dans une image, même si ce pattern a légèrement changé, puis de boucler sur cette opération pour traquer effectivement un pattern quelconque en entrée de la boucle. Il existe une architecture particulière de réseau qui exécute cette tâche: les réseaux siamois [10] et le région proposal network [11] ( un type de fast R-CNN [7] comme le backbone de YOLO ) formant le réseau SiamRPN [12], cf fig 5. Un réseau siamois est composé de deux réseaux convolutifs classiques, ici d'architecture AlexNET [13], entraîné en parallèle sur une fonction de coût qui compare les deux sorties, un réseau prendra en entrée le pattern de pixels à traquer et l'autre l'image dans laquelle faire le tracking. Les sorties des deux réseaux convolutifs nourrissent ensuite un réseau proposeur de région (RPN), qui, à partir des features map générées via l'image, va créer des ancrages de tailles différentes où un objet se trouvent potentiellement,( cf fig 4 ). A partir de ces boxes de tailles différentes et des features maps du pattern, le RPN [11] va estimer pour chaque ancre la probabilité que le pattern soit dedans et donc fournir une bounding box pour le pattern dans l'image. Le RPN [11] est composé d'un classifieur et d'un régresseur, le classifieur détermine la probabilité qu'une proposition, ancre, possède l'objet cible. Le régresseur estime les coordonnées des propositions. On choisit finalement la box la plus probable. On peut ensuite réutiliser le réseau avec l'image suivante et un nouveau pattern, le contenu

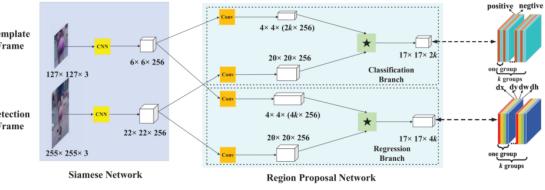


Fig. 5. SiamRPN, Crédit: [12]

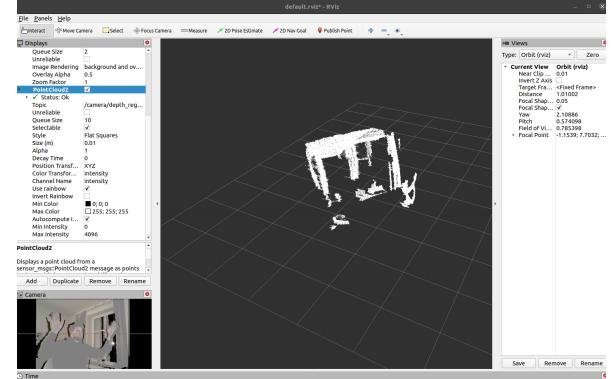


Fig. 6. Présentation de la caméra de profondeur(Xtion) sous rviz

de la bounding box précédente. Le réseau est entraîné sur une base de données de tracking très connue, le Youtube Bounding-Boxes DataSet qui fournit un très grand nombre d'échantillons d'objets traqués par des bounding boxes.

Le réseau est particulièrement performant et atteint 180 FPS sous GPU avec CUDA. Sur CPU en concurrence avec d'autres processus on ne dépasse jamais 0.25 secondes par images.

### III. MÉTHODE

#### A. Utilisation d'une caméra stéréoscopique

La caméra de tête du robot est une caméra stéréoscopique, c'est-à-dire qu'elle fournit une carte de profondeur (mappage des pixels sur des variations de gris, cf fig 6) associée à l'image RGB qu'elle renvoie. A partir de la carte de profondeur précédente il faut déterminer la position des pixels dans l'espace cartésien réel. Une première méthode consiste en un étalonnage manuel de la caméra, soit en mesurant les angles d'ouvertures de la caméra, pour associer position dans l'espace et pixels. Cependant ce n'est pas une méthode précise du à la nature des mesures à effectuer, se répercutant dans le tracking. Une nouvelle méthode réglant ce problème est l'utilisation de la matrice pixels-Transformation dans l'espace, fournis directement par la librairie PCL<sup>4</sup>. La précision de tracking a donc bien augmenté passant de +15cm à +4cm (cf résultats).

#### B. Structure du programme

L'architecture du système de contrôle du robot s'organise autour de ROS, pour robot operating system, ROS est

<sup>4</sup><https://pointclouds.org/>

constitué d'un cœur qui reçoit et transmet des informations à des nœuds connectés à des capteurs, moteurs, caméra, programmes... via des topics et des services. Dans le cadre de la téléopération par computer vision le programme utilise l'architecture de ROS pour permettre au réseau de neurones écrit en python de communiquer avec les drivers de la caméra stéréoscopique du robot et avec son contrôleur écrit en C++.

La caméra stéréoscopique envoie au cœur de ROS( cf Fig. 7) des données: l'image RGB et la carte de profondeur. Un premier nœud, tracking\_from\_cam utilise le réseau SiamRPN pour suivre dans l'image RGB la cible donnée par l'utilisateur. Cette cible est transmis à un autre noeud depth\_pcl qui vas déterminer sa position dans l'espace par rapport à la caméra puis à l'aide de la position de la tête, le noeud ramène la position de la cible dans le repère du robot, le dernier noeud, depth\_go\_first transmet au contrôleur le mouvement que le bras doit effectuer à 4 hz. En effet un des point qui a posé problème dans le développement du trackeur est que le contrôleur de localisation (tracking\_from\_cam) donne la position à l'instant même où le réseau profond a fini le traitement, or ce temps n'est pas une constante et, sur l'ordinateur utilisé pour faire tourner le système, varie entre 0.17s et 0.25s. Si le programme n'est pas cadencé, ceci provoque l'envoi de points à traquer à un rythme non contrôlé. Si le robot utilise le tracking mode, mode qui se déplace instantanément à un point donné, cela ne devrait pas poser souci. Cependant pour ne pas que le bras accélère trop vite, le point à rejoindre en tracking mode doit être à moins de 2 cm du précédent, une précision que n'offre pas le réseau ( cf résultats ). Dans cette configuration le programme bloque donc régulièrement et n'offre pas un tracking suffisant. Un autre mode de déplacement existe, le trajectory mode, il permet de générer une trajectoire de plusieurs points et d'une durée donnée entre deux points. Ainsi cette fréquence de 4hz fut donc choisie pour stabiliser le tracking du robot, en effet le traitement par le réseau de neurones ne prenant jamais plus de 0.25 secondes, il y au moins 4 points toutes les secondes. Et en générant via le trajectory mode des trajectoires de 0.25 secondes toutes les 0.25 secondes avec l'assurance d'avoir un nouveau point à rejoindre cette méthode permet d'assurer fluidité et fiabilité dans le tracking.

Une méthode permettant de prouver que le lien s'établit entre le traceur et le contrôleur fut de simuler le programme sur Gazebo ( cf Fig. 8 ) pour vérifier qu'il n'y aurait pas de destruction sur le robot.

#### IV. NOUVELLE MÉTHODE

##### A. Méthode 1

Cependant dans le cadre des expérimentations un problème bien connu en vision en robotique c'est rapidement présenté: l'occlusion, en effet dans certaine configuration le bras finit par s'interposer entre caméra et objet ( cf Fig.9, Fig. 10 ).

Une première méthode élégante, car originale, imaginée pour résoudre ce problème fut de tirer profit du système de collisions du contrôleur pour faire éviter la zone de champ

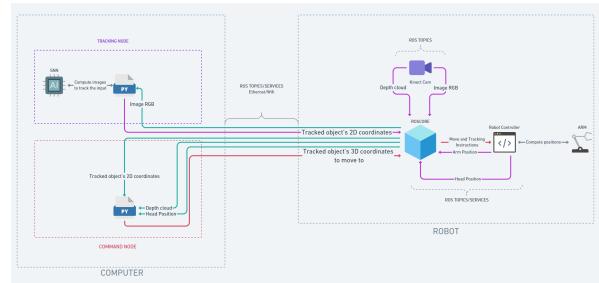


Fig. 7. Architecture ROS

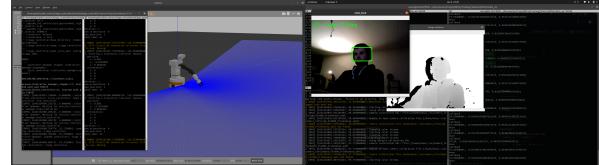


Fig. 8. Programme simulé sous Gazebo

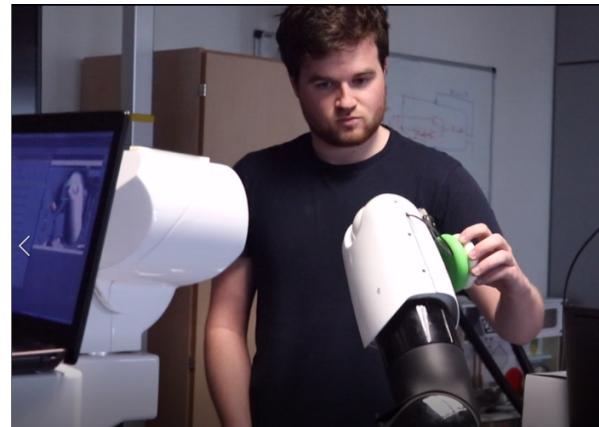


Fig. 9. Occlusion lors du tracking

de vision au bras du robot. En établissant de nouvelles tâches de collisions (cf Fig. 11 ) avec un poids faible suivant une fonction de coût en exponentielle (de multiples sphères) [2] pour modéliser le champ de visions.

##### B. Résultats 1

Cependant lors de la simulation tout ne se passe pas comme prévu et le bras ignore les sphères

- Augmentation du poids de la tâche mais le bras ne vas plus bouger.
- Diminution du poids et augmentation du rayon d'action des sphères. Mais cela ne marche pas non plus, le bras se comporte de façon étrange et refuse de contourner les sphères.

Une idée est que le contrôleur ne peut pas voir "le futur" ainsi malheureusement il n'y a pas de gradient qui indique au contrôleur du robot que faire une rotation du coude serait bénéfique dans le futur et permettrait de réduire le coût de la tâche de collisions, le contrôleur voit juste que sur le moment entamer une rotation ne réduit pas plus le coût de la tâche qu'autre chose et donc ne le fait pas.

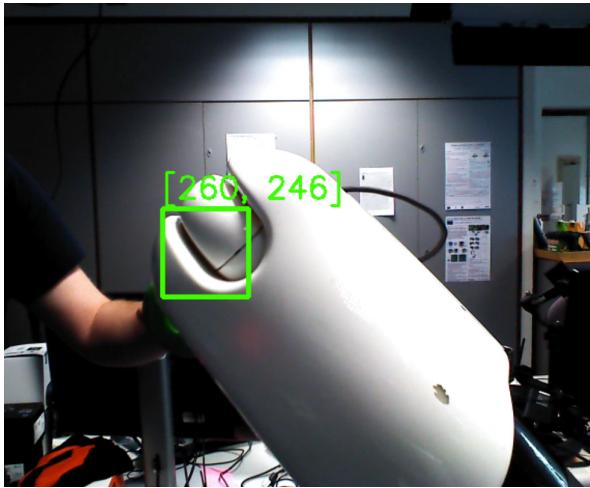


Fig. 10. occlusion vu par la caméra

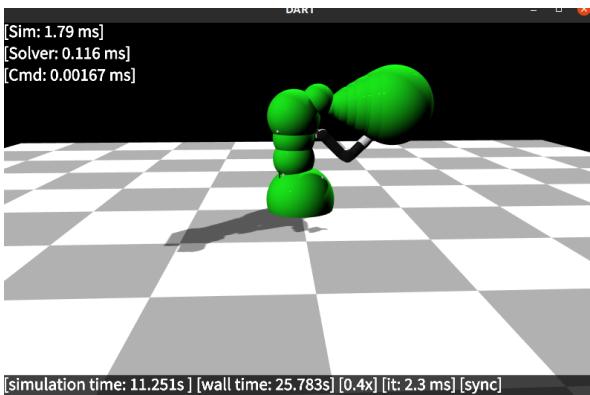


Fig. 11. Sphères de collision dans le simulateur DART

### C. Méthode 2

une autre solution à l'occlusion, plus simple, est l'utilisation de deux caméras. Ceci qui nécessite de synchroniser deux traqueurs et de gérer leurs position dans l'espace afin de choisir la bonne caméra pour le tracking. La nouvelle caméra est placée sur la tête du robot, 50 cm au-dessus et avec un angle de 40 degrés. Il faut à nouveau ramener la position de l'objet traqué du repère de la caméra vers le repère du robot. Ici on effectue d'abord une transformation du repère orienté à 40 degrés R2 vers celui parallèle à celui du robot R1, ensuite on revient dans le repère cartésien du robot R0 en sommant les coordonnées de la tête aux coordonnées précédentes. La transformations de R2 vers R1 est la suivante:

$$\begin{aligned} x_1 &= y_2 \sin(40) + x_2 \cos(40) \\ y_1 &= y_2 \cos(40) - x_2 \sin(40) \end{aligned}$$

Une fois les coordonnées émises par les deux caméras obtenues et ramenées dans le repère cartésien du robot on peut les comparer. Et ici il y a un trop grand écart entre les deux, il y a occlusion et on peut choisir la caméra du haut.

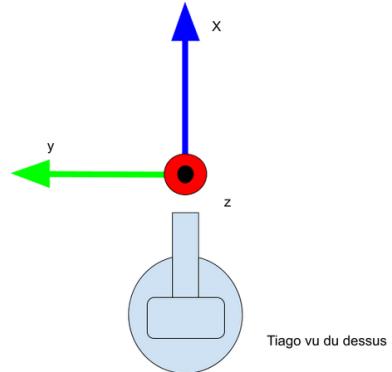


Fig. 12. Repère cartésien du Tiago

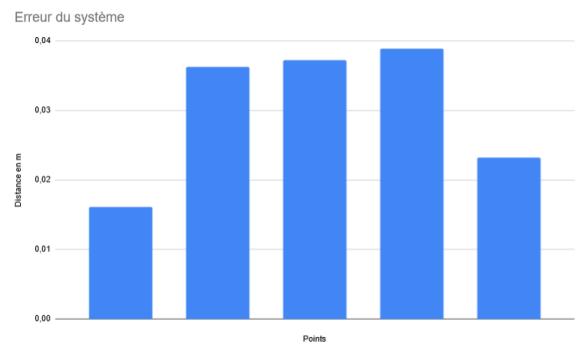


Fig. 13. Erreur du système

## V. RÉSULTATS

Afin de déterminer la précision du système de tracking plusieurs expériences de mesures ont été effectuées. Pour effectuer ces mesures un objet a été placé à plusieurs endroits de positions différentes et connues dans l'espace cartésien du robot ( cf Fig. 12 ).

L'erreur du système ( cf Fig. 13 ) correspond à la distance entre le point connu à l'avance et le point atteint par le bras, end-effector, selon le contrôleur. L'erreur sur le système se trouve donc entre 2 et 4 cm, plus souvent proche de 4cm. Cette erreur peut s'expliquer par plusieurs facteurs ( cf Fig. 14 ) :

- L'erreur de localisation correspond à la distance entre le point connu à l'avance et le point déterminé par le réseau de neurone.
- L'erreur de positionnement correspond à la distance entre la commande passée par le nœud utilisant le réseau de neurones et le point déterminé par le bras.

On note que la somme des erreurs ne correspond pas à l'erreur du système ( cf Fig. 14 ), il y a un effet de compensation entre les deux erreurs. Comme on le note sur le graphique une partie de l'erreur vient de l'erreur de positionnement. En effet, il est possible que le bras n'atteigne pas exactement la position qui lui est demandée, car le contrôleur prend en compte toutes les tâches, et en fonction des poids qui

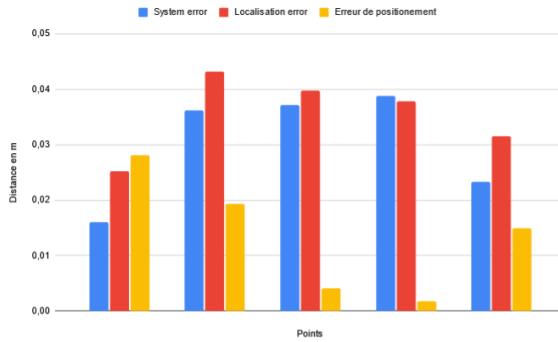


Fig. 14. Sourçs d'erreurs

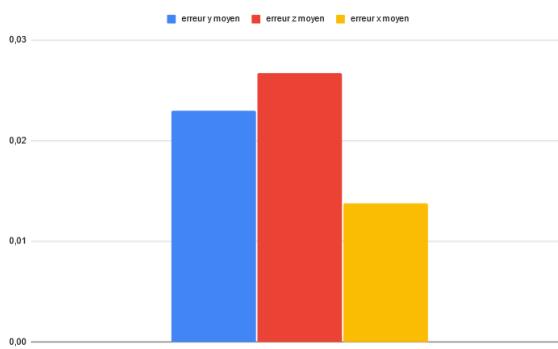


Fig. 15. Erreur moyenne de la caméra xtion dans le repère cartésien Fig.12

leurs sont affectées ne va pas forcément atteindre la cible initiale. Cependant la majeure partie de l'erreur vient de l'erreur de localisation. Cette erreur est due à l'erreur de la caméra stéréoscopique, sur la conversion pixels vers espace cartésien. Pour obtenir des précisions sur cette erreur, des mesures ont été effectuées sur les 3 axes de la caméra

## VI. CONCLUSION

L'erreur trouvée sur le système montre que le programme fonctionne raisonnablement. Cependant l'erreur de 3cm reste assez grande pour un système robotique. Dans l'état actuel le système est potentiellement plus précis, car les mesures sur la position réelle de l'objet utilisé pour les mesures ont été réalisé avec un mètre il y a une fortes incertitudes sur ces mesures, pour déterminer au millimètre près ces positions il faudrait utiliser un système de motion tracking. Si le tracking est fonctionnel, il reste maintenant à faire interagir le bras avec l'environnement traqué.

stéréoscopique à différents points connus ( cf Fig. 15 ).

On peut noter que l'erreur sur x, soit la profondeur est nettement plus faible que les autres. En effet si la carte de profondeur de la caméra est précise, le problème reste le même que cité précédemment, les coordonnées y et z sont difficile à déterminer, et si la conversion de PCL est plus précise qu'un étalonnage manuel, il y a toujours une erreur assez importante qui se répercute sur la localisation de la cible.

## REFERENCES

- [1] D. Roetenberg, H. Luinge, and P. Slycke, "Xsens mvn: Full 6dof human motion tracking using miniature inertial sensors," *Xsens Motion Technologies BV, Tech. Rep.*, vol. 1, 2009.
- [2] E. Dalin, I. Bergonzani, T. Anne, S. Ivaldi, and J.-B. Mouret, "Whole-body teleoperation of the Talos humanoid robot: preliminary results," in *ICRA 2021 - 5th Workshop on Teleoperation of Dynamic Legged Robots in Real Scenarios*, Xi'an / Virtual, China, May 2021. [Online]. Available: <https://hal.inria.fr/hal-03245005>
- [3] K. Bouyarmane and A. Kheddar, "On weight-prioritized multitask control of humanoid robots," *IEEE Transactions on Automatic Control*, vol. 63, no. 6, pp. 1632–1647, 2017.
- [4] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integrations (SII)*, 2019.
- [5] J. Carpentier and N. Mansard, "Analytical derivatives of rigid body dynamics algorithms," in *Robotics: Science and Systems*, 2018.
- [6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015. [Online]. Available: <https://arxiv.org/abs/1506.02640>
- [7] R. Girshick, "Fast r-cnn," 2015. [Online]. Available: <https://arxiv.org/abs/1504.08083>
- [8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013. [Online]. Available: <https://arxiv.org/abs/1311.2524>
- [9] J. Liu, J. Sun, and S. Wang, "Pattern recognition: An overview," *IJCSNS International Journal of Computer Science and Network Security*, vol. 6, no. 6, pp. 57–61, 2006.
- [10] G. R. Koch, "Siamese neural networks for one-shot image recognition," 2015.
- [11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: <https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf>
- [12] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8971–8980.
- [13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>