Projet Hexapode
Utilisation d'un robot pour le dessin

Pierre-Louis PHAN, Ghani KISSOUM, Yassine FENDI, Youcef BEN AKMOUME, Badr HANNAOUI
M2-SPI, Sorbonne Université

***Résumé*— Cet article décrit les différents travaux menés par notre équipe au cours du projet intégratif de M2 SDI de l'Université Pierre et Marie Curie. L'idée est de reprendre un robot hexapode réalisé l'année précédente, de l'étudier, l'améliorer et de l'adapter afin d'en faire un robot qui transforme une image donnée par l'utilisateur en une trajectoire de référence tout en y intégrant un outil de dessin. Notre robot artiste est capable de se repérer dans l'espace grâce à un système de visualisation composé d'une caméra et de marqueurs, et de se déplacer à l'aide de servomoteurs. Ceci dans le but de pouvoir reproduire, en toute autonomie, une image donnée en la dessinant sur une feuille au sol par l'intermédiaire d'un feutre et d'un programme de traitement d'image. Un des enjeux de ce projet est de proposer une entrée au concours RobotArt [1], qui consiste simplement à concevoir un robot capable de produire des œuvres de manière autonome.**

I. INTRODUCTION

A. État de l'art des travaux sur les hexapodes

Un robot hexapode est un robot à six pattes, inspiré de la nature. Plusieurs modèles de robots hexapodes existent, que ce soit à des fins d'apprentissage ou bien pour le développement et la recherche. Ils sont dotés d'une autonomie suffisante pour circuler dans l'environnement et pouvoir effectuer les tâches demandées. Leur versatilité, due à un nombre élevé de degrés de libertés, les rend très adaptés pour circuler dans des environnements complexe tout en gardant une stabilité optimale. Ils ont connu un succès croissant durant les dernières décennies auprès de quiconque ayant un attrait pour la robotique, professionnel ou amateur. C'était devenu un symbole de l'intégration des connaissances robotiques, et récemment la conception de machines robotisées a connu un grand saut, alliant un design et une fabrication assez efficace pour nous permettre des applications plus concrètes.

Une quantité importante de travaux sur le concept de robot hexapode ont été réalisés. Le Jet Propulsion Laboratoryde la NASA a conçu un hexapode versatile pour la maintenance de panneaux solaire orbitaux [2] ; il devrait être capable de travailler minutieusement sur de petites pièces d'assemblage. Le concept d'hexapode a été fortement simplifié par des chercheurs de l'University of Michigan : un robot dont les pattes flexibles ne sont dotées que d'un seul moteur chacune, et qui se révèle extrêmement agile sur terrain accidenté [3].

En ce qui concerne les travaux amateurs, un projet de "Tracking and Following" de David Henderson [4] a permis à un hexapode de suivre des objets de couleurs avec des mouvements très proches de ceux d'un insecte. Un autre travail de Kevin Ochs [5] stabilise l'hexapode horizontalement même sur un relief complexe, ce qui lui permet de faire du SLAM (Simultaneous localization and mapping) en extérieur. Mieux encore, Kåre Halvorsen [6] a conçu un petit robot hexapode doté d'une initiative propre, et capable de repérer les obstacles, de les anticiper pendant le déplacement.

B. Description du robot utilisé

Pour notre projet, nous utilisons l'hexapode à dix-huit axes construit lors d'un projet précédent (Fig. 1). Il est inspiré d'un modèle existant, le PhantomX. Les axes sont motorisés par des servomoteurs Dynamixel pouvant être pilotés en position avec une résolution de 0.29 degrés°, et en vitesse avec une précision de 0.111rpm. Il est doté de deux caméras 5 mégapixels, d'un module ultrason, et d'un capteur de pression au bout de chaque patte. Une carte Arduino acquiert les données des capteurs, et deux cartes Raspberry Pi offrent une puissance de calcul disponible pour un grand nombre d'applications, en plus de piloter les servomoteurs par les drivers adaptés.

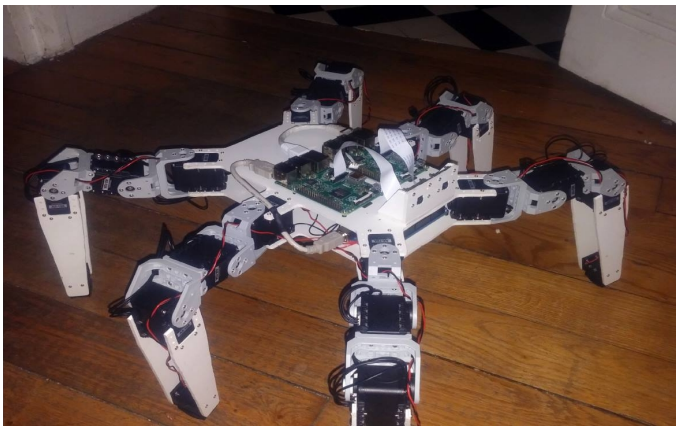


Fig. 1: Le robot utilisé

Dans ce travail, sa tâche primaire sera de tracer un dessin sur une surface plate en se basant sur une image donnée en entrée, ou ce qu'il voit à sa caméra.

L'objectif principal est donc de reproduire, sur une feuille fixée au sol, n'importe quelle image donnée en entrée. N'étant doté que de feutres, le dessin obtenu est composé de plusieurs tracés de même épaisseur ; nous avons donc décidé de lui faire reproduire seulement les contours de l'image. Plus précisément, l'œuvre finale est un ensemble monochrome ou polychrome, de tracés rectilignes ou courbes, composés en fait de points reliés entre eux. Ces tracés

suivent des contours obtenus par un traitement sur l'image d'origine, la complexité du résultat dépend donc directement de la complexité de l'image.Les feutres sont fixés aux pattes, et la précision des moteurs est suffisante pour obtenir une reproduction fidèle de l'image. Il effectue des reproductions de taille assez grande (de l'ordre de 2m x 2m) afin d'avoir encore plus de précision relativement à la taille de l'image.

II. PLANIFICATION

L'intérêt d'un robot mobile est d'avoir un champ d'action potentiellement infini : en se déplaçant, il déplace sa zone de travail avec lui. L'hexapode est donc capable de réaliser une œuvre plus grande que lui. Cela implique de se déplacer, ce qui implique à son tour de pouvoir se repérer précisément. On se retrouve donc avec une multitude de tâches, à organiser correctement pour atteindre le but final. Ci-dessous, une présentation globale de la planification adoptée. Elle sera suivie des descriptions plus en détails, étape par étape.

On commence par le traitement d'image, qui génère les différents tracés à suivre. Ce traitement s'effectue avant que le robot ne commence à bouger. C'est la suite de plusieurs opérations : détection de contours et réduction de leur épaisseur ; on obtient alors un nuage de points (x,y) dans le plan du sol. Puis un algorithme de notre conception relie ces points entre eux, pour en faire des trajectoires : des listes de points. C'est cette liste de "listes ordonnées de points" que l'on donne en consigne à l'hexapode, son but étant de les relier au feutre.

Pour plus de précision dans son tracé, le robot ne dessine que lorsqu'il est immobile (et toujours dans la même pose). Il dessine dans la zone disponible, puis il se déplace pour dessiner ailleurs. Pour un gain d'efficacité, le robot commence par les zones où il aura le plus de dessin à faire : il repère, par recherche aléatoire, la zone où le maximum de points sont à sa portée. Pour s'y rendre, il effectue une suite de déplacements linéaires et de rotations. Nous avons opté pour la démarche classique pour les hexapodes : trois pattes au sol, trois pattes en l'air. Cependant beaucoup de facteurs perturbateurs influent sur le déplacement réel de l'hexapode.

Faute de pouvoir se déplacer précisément, il peut se localiser précisément. En faisant en sorte que la caméra ait toujours au moins un marqueur dans son champ de vision, l'angle de vue qu'il a sur le marqueur lui détermine sa position. La détection de marqueurs est réalisée par un algorithme préexistant, et une DLT permet de connaître la position du marqueur dans le repère du robot. Après des changements de repères, il connaît sa position (3 coordonnées, 3 angles) dans le repère absolu, et on recoupe cette information avec celle apportée par chaque marqueur visible.

De cette manière, même s'il n'a pas atteint exactement la position voulue, on peut déterminer exactement dans le repère du robot les coordonnées des tracés voulus. L'hexapode se met alors en position de dessin ; sa position au repos lui permet de lever jusqu'à trois pattes sans tomber. En se servant d'un modèle géométrique, chaque patte peut être asservie par la position (x,y,z) de son extrémité, ou alors de l'extrémité de son outil de dessin. Il peut ensuite relier entre eux les points appartements au même tracé, et effectuer toutes les portions de tracés atteignables.

Ce processus est répété, jusqu'à ce que tous les points aient été dessinés.

III. TRAITEMENT D'IMAGE

A. Détecter des contours

Le robot doit être capable de dessiner de manière autonome une image donnée, et ceci sans se perdre dans les détails inutiles de l'image. L'idée est donc de ne détecter que les parties significatives de l'image. De plus, le robot ne peut tracer que des lignes, donc la solution triviale est la détection des contours de l'image. Plusieurs méthodes existent ; nous avons choisi le classique détecteur de Canny [7] pour les raisons suivantes :

- Bonne détection : un faible taux d'erreur dans la signalisation des contours
- Bonne localisation : une minimisation des distances entre les contours détectés et les contours réels
- Clarté de la réponse : une seule réponse par contour et pas de faux positifs

(a) Avant (b) Après

Fig. 2: Application de la détection de contours de Canny

L'algorithme de Canny se compose de plusieurs parties :

- Réduction du bruit : Garantie la clarté de la réponse en éliminant les pixels isolés évitant ainsi d'obtenir de faux positifs. Le filtre utilisé est de taille (5,5).
- Gradient d'intensité : L'image lissée est ensuite filtrée par un noyau de Sobel selon la direction horizontale et verticale afin de trouver les directions horizontale (Gx) et verticale (Gy). Ce qui permet de retrouver le gradient d'intensité du contour et son orientation :

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = tan^{-1}(\frac{G_y}{G_x})$$

- Suppression des non-maxima : Arrivant à ce stade, chaque pixel a une forte intensité, ce qui augmente sa possibilité d'être un point contour , pour trancher, seuls les points représentant des maximas locaux sont conservés.
- Seuillage des contours (seuillage par hystérésis) : Ici on décide si un pixel appartient au contour ou non. Cela nécessite deux seuils : un haut et un bas, qui seront comparés à l'intensité du gradient de chaque

point. La valeur du seuil minimal est utilisée pour assurer la liaison entre les contours afin de ne pas avoir de grandes coupures ; et le seuil maximal est utilisé pour détecter les parties les plus significatives.

À ce stade, on obtient une liste de points appartenant au contour (Fig. 2b).

B. *Générer les trajectoires*

Notre objectif étant de tracer les contours de l'image, et non pas l'image en pointillés, il faut transformer les points en trajectoires ; c'est à dire une liste ordonnée de points. Pour cela, nous avons conçu un algorithme qui génère des trajectoires sur la notion de voisinage d'un point.

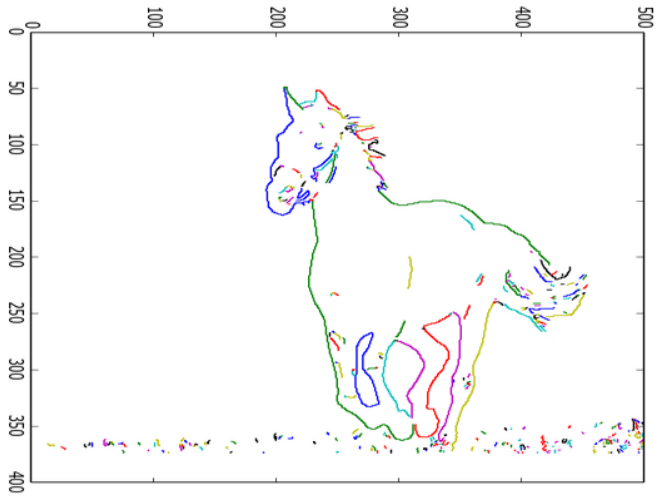
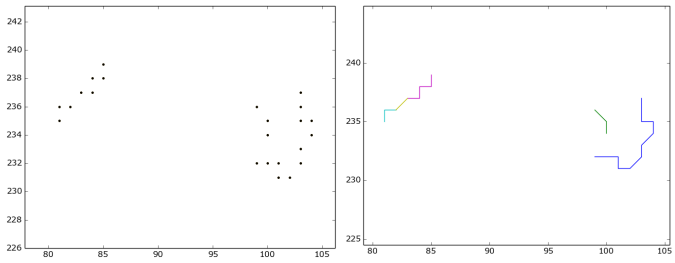


Fig. 3: Une visualisation des trajectoires calculées sur le nuage de points de la Fig. 2b

Chaque point (i, j) a 8 voisins potentiels $(i \pm 1, j \pm 1)$: .Deux voisins se retrouveront adjacent dans la trajectoire. On parcourt l'image, et pour chaque point rencontré, on compte son nombre V de voisins (Fig. 13) :

- V = 0 : On ouvre une nouvelle trajectoire qui n'est composée que de ce point, et on la referme.
- V est impair : Ce point est à une extrémité ; on ouvre une nouvelle trajectoire, qui se propage de voisin en voisin. Au fur et à mesure, on rajoute dans la liste les points trouvés : on les "classe" dans cette trajectoire, et on les enlève de l'image. Si on tombe sur un point à plusieurs voisins (c'est à dire : si la trajectoire se sépare), on prend au hasard l'un des chemins, et les autres seront traités plus tard dans d'autres trajectoires. Si on tombe sur un point à 0 voisin, on referme la trajectoire.
- V est pair : Ce point est en milieu de trajectoire ; on l'ignore car il sera traité par le cas ci-dessus.

L'algorithme continue ainsi jusqu'à avoir traité tous les points du contour. À la fin, il reste des boucles, car elles n'ont pas d'extrémité par laquelle commencer la trajectoire. On commence alors à un des points au hasard, pour "casser" la boucle.



(a) Avant (b) Après

Fig. 4: Application de l'algorithme de génération de trajectoires sur un nuage de points "test"

IV. DÉPLACEMENT

A. *Modes de démarche*

Le robot dessine sur place, pas en se déplaçant. Il dessine dans la zone accessible, puis se déplace pour dessiner ailleurs.Dans l'état de l'art, il existe différents modes de déplacements [8].

Le plus répandu est le mode tripode (Fig. 5), qui divise les pattes en deux groupes de trois pattes. Les jambes d'un même groupe sont levées, avancées, et abaissées à l'unisson ; et pendant qu'un groupe de pattes se déplace en l'air, l'autre est au sol et déplace le corps du robot. Puisqu'à tout moment trois pattes sont au sol, cette approche est à la fois «statiquement» et «dynamiquement» stable.

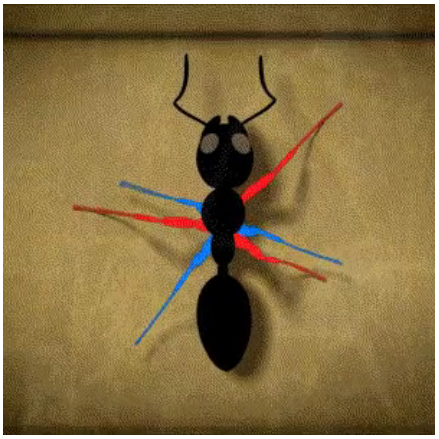


Fig. 5: Démarche tripode : c'est la même démarche que celle adoptée par les insectes

Un autre mode de déplacement : le mode 'wave'. Toutes les pattes d'un côté sont déplacées successivement en commençant par la jambe à l'arrière, et c'est ensuite répété de l'autre côté. Puisqu'une seule patte est élevé à la fois, les 5 autres étant au sol, l'hexapode est toujours dans une posture très stable. Malheureusement la cadence de ce mode là ne peut pas être accélérée, au risque de chevaucher les phases de suspension. Ce mouvement est analogue à celui des mille-pattes.

B. Mise en œuvre

Nous avons opté pour la démarche tripode, à la fois pour la marche (déplacement rectiligne) et la rotation (déplacement angulaire). Dans les deux cas, le principe est le même :

- Effectuer, avec les extrémités des trois pattes levées, le mouvement souhaité
- Effectuer, avec les extrémités des trois pattes au sol, le mouvement opposé
- Poser les pattes levées, lever les pattes posées, et réitérer

En réalité les extrémités des pattes au sol restent immobiles, à condition d'effectuer le même mouvement ; par réaction, c'est le corps du robot qui effectuera le mouvement opposé.

Pour la marche rectiligne, les extrémités des pattes levées parcourent une distance $D > 0$ sur l'axe $\overrightarrow{X}$ (axe de roulis du robot), tandis que les pattes au sol parcourent une distance $-D$. Soit (x_0, y_0, z_0) la position de l'extrémité de chaque patte au repos, on effectue alors simultanément pour les pattes respectivement levées et posées, les mouvements suivants :

$$(x_0-\frac{d}{2}, y_0, z_0+h) \longrightarrow (x_0+\frac{d}{2}, y_0, z_0+h) \longrightarrow (x_0+\frac{d}{2}, y_0, z_0)$$

$$(x_0+\frac{d}{2}, y_0, z_0) \longrightarrow (x_0-\frac{d}{2}, y_0, z_0) \longrightarrow (x_0-\frac{d}{2}, y_0, z_0+h)$$

Pour la rotation, on commence par placer les 6 extrémités des pattes sur un même cercle centré en $(0, 0, 0)$, l'origine du repère du robot (Fig. 6). Ensuite, de manière analogue à la marche, les extrémités des pattes levées parcourent un angle θ autour de l'axe $\overrightarrow{Z}$ (axe de lacet du robot), tandis que les pattes au sol parcourent un angle $-\theta$. Le signe de θ détermine le sens de rotation

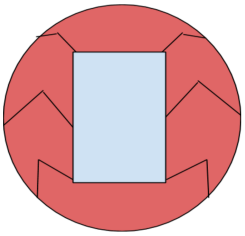


Fig. 6: Position prise avant d'entamer la rotation

Ainsi pour atteindre une position cible, l'hexapode commence par pivoter pour "viser" cette position, puis marche directement devant lui sur la distance souhaitée. L'incertitude constatée durant le déplacement réel du robot, due au jeu dans les moteurs et au glissement involontaire des pattes, a pour conséquence une erreur sur la position finale du robot. Cette erreur se révèle d'autant plus grande que le déplacement souhaité est grand : pour la même erreur d'angle en visant la position cible, plus la distance à parcourir est grande, plus le robot s'écarte de la ligne prévue. Or, après son déplacement, il s'est tout de même significativement rapproché de la position cible, et peut connaître sa position réelle avec précision (voir section V). Il suffit donc de réitérer le processus, et les mêmes incertitudes auront moins d'effet sur l'erreur finale. Nous avons constaté qu'il est parfois nécessaire de répéter ce processus une troisième fois pour atteindre une précision de l'ordre du centimètre.

Pour ces déplacements, il ne suffit pas d'indiquer en commande les points initiaux et finaux. En effet on souhaite qu'une extrémité de patte progresse linéairement dans l'espace 3D réel. Si une patte progresse linéairement dans l'espace de configuration (q_1, q_2, q_3), alors l'extrémité de cette patte suivra une trajectoire courbée : les actionneurs ne produisent pas des mouvements de translation. Ainsi, pour qu'une extrémité aille du point P_1 au point P_2, il faut donner en consigne successivement les points d'interpolation $(1-x).P_1 + x.P_2$, où x parcourt $[0, 1]$ avec un pas réglable.

Pour asservir directement l'extrémité de patte dans l'espace 3D, nous nous servons du modèle géométrique inverse (voir section VI-B).

C. Recherche d'une zone d'intérêt

Le robot peut donc se déplacer précisément sur toute la surface à dessiner, mais un algorithme de sélection de zone est indispensable pour décider où se déplacer.

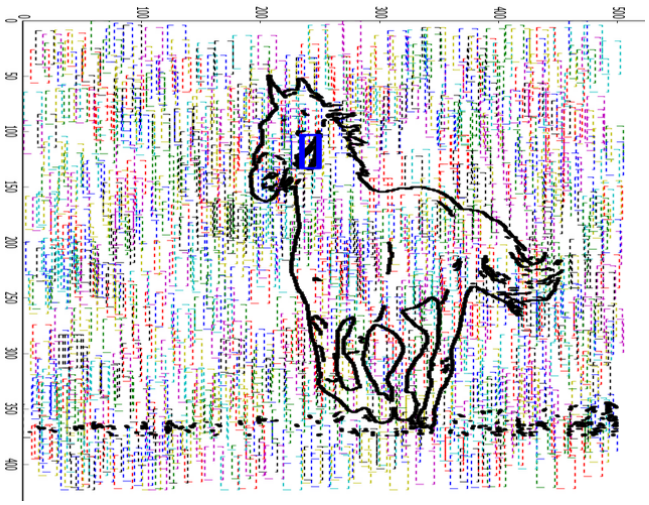


Fig. 7: Résultat de la recherche d'une zone d'intérêt. Le rectangle montre une des zones contenant le plus de points. Vu son caractère probabiliste, le résultat peut différer, mais sans trop varier.

Ce dernier se base sur une recherche aléatoire : On applique au nuage de points un masque, correspondant au champ d'action fixe du robot (voir section VI-C). Ce masque parcours aléatoirement le nuage de points, et compte simplement le nombre de points inclus. On garde la zone avec le nombre de points maximal, définie par son point d'application et son angle d'orientation. On montre Fig. 7 le résultat de l'algorithme. En pointillés : tous les cas étudiés, et en gras la zone retenue. Il est à noter que la zone d'application du robot est représentée ici comme un rectangle, mais le masque appliqué est en réalité de la forme du champ d'action du robot.

V. LOCALISATION PAR VISION

A. Contexte

La connaissance de la position et de l'orientation du robot au moment de la réalisation du dessin est indispensable, notamment puisque le robot doit réaliser des déplacements au cours de sa tâche. Une première solution pourrait être un calcul incrémental des déplacements et rotations effectuées par le robot pour déterminer sa position, néanmoins cette solution n'est plus envisagée à cause des imprécisions sur le modèle géométrique du robot, et les phénomènes de glissement des pattes. Une meilleure solution consiste donc à exploiter l'information visuelle fournie par la caméra du robot. C'est une solution à faible coût largement utilisée en robotique [9] [10].

Un système de localisation par la vision utilise une ou deux caméras, et doit détecter certains points caractéristiques de l'environnement [11]. Si le système de vision n'emploie qu'une seule caméra, alors la position du robot peut être calculée avec deux images prises par la caméra lors du déplacement du robot. Pour pouvoir calculer la position du robot à chaque instant, et avec une seule capture de la caméra, il faut utiliser des marqueurs facilement détectables, et différentiables par un algorithme de détection.

B. Détection de marqueurs ArUco

Nous avons employé des marqueurs particuliers qui peuvent facilement être détectés par un algorithme de traitement d'images. Ces marqueurs possèdent des codages particuliers qui permettent de les différentier.

La bibliothèque ArUco fut initialement créée pour des applications de réalité augmentée ; elle regroupe des algorithmes qui permettent de générer et de détecter des marqueurs de 7x7 carrés, noirs ou blancs (Fig. 8) [12]. Un système de codage issu de la théorie d'information permet d'assigner un identifiant unique pour un marqueur et ce dernier ne pas être obtenu par la rotation d'un autre marqueur, ainsi, les risques de confusion et de fausses détections sont largement réduits [13].

C. Calcul de la position du robot

Les marqueurs ayant des identifiants différents à la détection, ils peuvent être fixés à des endroits déterminés dans l'environnement du robot, ainsi la position de chaque marqueur est bien connue dans le repère monde. On décrit ci-dessous la méthode de calcul de la position et de l'orientation du robot après avoir détecté un des marqueurs :

En connaissant les paramètres intrinsèques de la caméra regroupés dans une matrice K de dimension 3x3, et connaissant également la position de chaque marqueur dans le repère monde, on peut déterminer la pose de la caméra par rapport au repère monde, et donc déterminer la position et l'orientation du robot.

La projection d'un point de l'environnement dans une image peut être décomposée deux transformations successives :

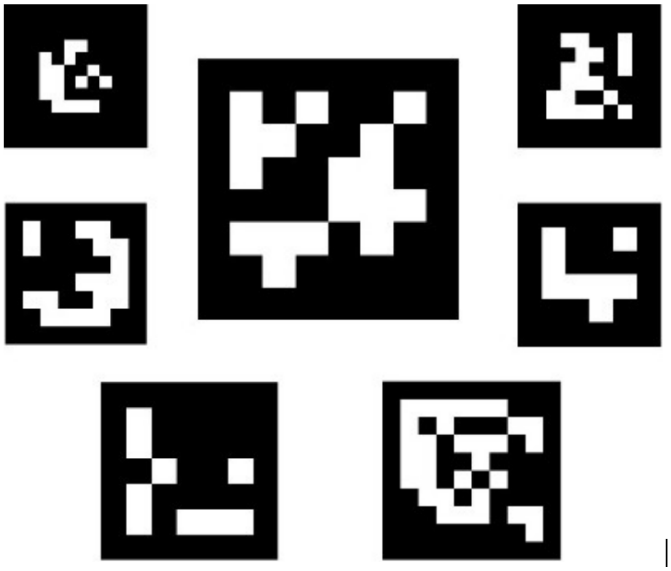


Fig. 8: Quelques marqueurs ArUco

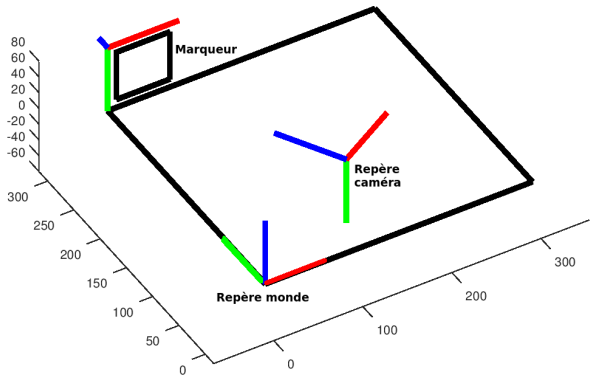


Fig. 9: Visualisation des transformations : Robot → Caméra → Marqueur → Monde

— Du repère monde, vers le repère de la caméra : une transformation M composée d'une rotation R et d'une translation T.
— Du repère caméra vers l'image : une transformation K qui regroupe les paramètres intrinsèques de la caméra.

Un point (x, y, z) dans le repère monde se transforme dans la photo à un point (u, v), selon la formule suivante :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \times [RT] \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

où u et v sont des entiers, correspondant à des coordonnées de pixels. Dans notre cas, on connaît (x, y, z) et K, et on a mesuré (u, v). On détermine alors la transformation $[RT]$ par la méthode de la DLT, qui inverse l'équation à l'aide d'une décomposition des données en valeurs singulières [14].

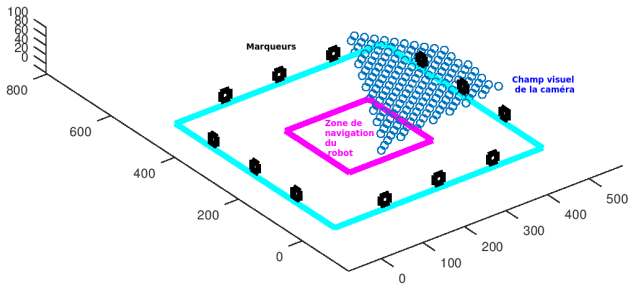


Fig. 10: Disposition des marqueurs dans l'environnement : nous en avons fixé suffisamment pour qu'au moins un marqueur soit visible par la caméra à tout moment

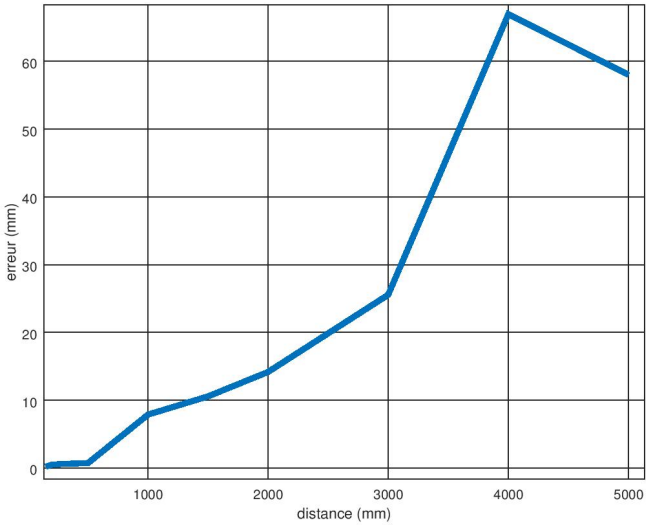


Fig. 11: Erreur statique sur la détermination de la composante en Z, en fonction de la valeur de Z

D. Évaluation des résultats

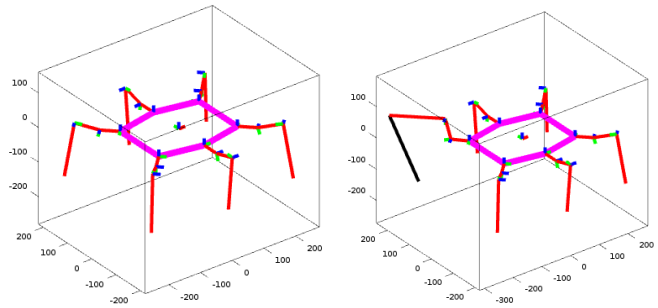
Pour évaluer les performances du système de localisation par la vision, nous avons réalisé une expérience dans laquelle on compare la position de la caméra (valeur de la composante en $\overrightarrow{Z}$ du vecteur translation obtenu) fourni par l'algorithme de localisation, et sa position réelle mesurée précisément. Étant limités par la contrainte d'espace et du matériel nécessaires pour réaliser l'expérience, nous n'avons pris en considération que la translation (et une seule composante de cette translation) sans vérifier l'erreur en rotation, beaucoup plus difficile à estimer. Les résultats de cette expérience sont montrés Fig. 11 où on trace l'évolution de cette erreur en fonction de la distance entre le robot et le marqueur. En s'éloignant, le marqueur représente moins de pixels sur la photo capturée, ce qui augmente l'incertitude. On constate, pour des distances de 15cm à 5m, que l'évolution de l'erreur est pire que proportionnelle à la distance par rapport à la caméra.

L'erreur maximale enregistrée sur la distance entre la caméra du robot et le marqueur est de 6.7 cm. Afin de réduire l'erreur sur l'estimation de la position du robot par ce système, l'algorithme calcule la position du robot par rapport à tous les marqueurs présents dans son champ visuel, et donne la moyenne de ces positions.

VI. DESSIN

A. Modèle géométrique direct

Pour la visualisation en 3D de l'effet des commandes données aux moteurs, il nous a fallu déterminer le modèle géométrique direct d'une patte du robot : exprimer les coordonnées du point (x, y, z) extrémité de la patte, en fonction des consignes (q_1, q_2, q_3) que l'on donne.



(a) Sans outil de dessin (b) Avec outil de dessin

Fig. 12: Visualisation de l'hexapode, en fonction des 18 commandes données aux moteurs

On a donc considéré 4 repères (Fig. 12a) :
- R_0 Repère Robot : Origine au centre du robot, $\overrightarrow{X}$ orienté à l'avant
- R_1 : Origine sur le premier axe de la patte, $\overrightarrow{X}$ orienté vers le deuxième axe
- R_2 : Origine sur le deuxième axe de la patte, $\overrightarrow{X}$ orienté vers le troisième axe
- R_3 : Origine sur le troisième axe de la patte, $\overrightarrow{X}$ orienté vers l'extrémité de la patte

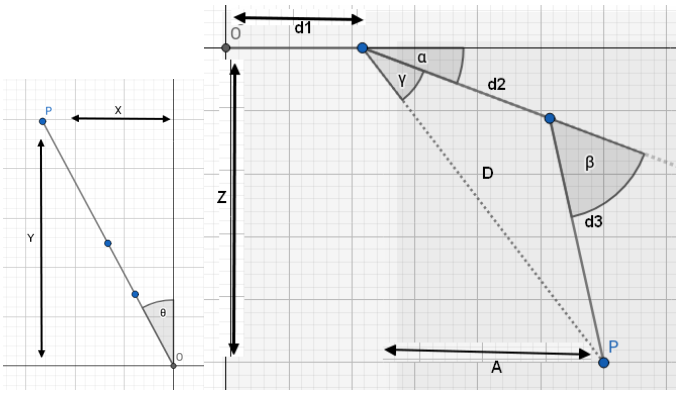
Les matrices de transformation M_i entre ces repères sont fonction des angles donnés en consigne. Soit $P = \begin{bmatrix} d_3 & 0 & 0 \end{bmatrix}^T$ les coordonnées du point extrémité de la patte, avec d_3 la longueur du dernier segment de patte, on a $P' = \begin{bmatrix} x & y & z \end{bmatrix}^T$ ses coordonnées dans le repère du robot par :

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = M_1 \times M_2 \times M_3 \times \begin{bmatrix} d_3 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Dans le cas où l'on dispose un outil dont l'extrémité est à une translation Tr en bout de patte, on part du point $P = \begin{bmatrix} Tr_x + d_3 & Tr_y & Tr_z \end{bmatrix}^T$ (Fig. 12b).

B. Modèle géométrique inverse

Et pour bien des aspects du travail effectué, nous nous sommes servis de l'inverse : déterminer les (q_1, q_2, q_3) à donner en consigne, pour atteindre un point (x, y, z) souhaité. Il n'est pas possible d'inverser simplement l'équation du modèle direct : il faut se servir de considérations géométriques.



(a) Calcul de θ (b) Calcul de α et β

Fig. 13: On détermine directement le modèle géométrique inverse

Pour l'angle 1 : $q_1 = \theta$
d'après Fig. 13a on a directement

$$\theta = 2 * \arctan(\frac{Y}{X + \sqrt{X^2 + Y^2}})$$

Puis l'angle 3 : $q_3 = \beta$
Seul cet angle détermine la distance D (cf. Fig. 13b) Le théorème d'Al-Kashi nous donne β :

$$D^2 = d_2^2 + d_3^2 - 2 * d_2 * d_3 * \cos(\frac{\pi}{2} - \beta)$$

Enfin l'angle 2 : $q_2 = \alpha$
β étant fixé, on doit maintenant orienter le bras vers P (cf. Fig. 13b) Le théorème d'Al-Kashi nous donne γ :

$$d_3^2 = D^2 + d_2^2 - 2 * D * d_2 * \cos(\gamma)$$

or, pour orienter le bras vers P, on veut :

$$\alpha + \gamma = 2 * \arctan(\frac{-Z}{A + \sqrt{A^2 + Z^2}})$$

ce qui nous donne α

On peut également intégrer l'outil de dessin, et asservir l'extrémité de cet outil. Sur Fig. 12, on constate que le calcul est correct : dans les deux cas, l'extrémité considérée se trouve au même point.

C. Détermination du champ de travail

Une fois que le robot se trouve à une position acceptable, proche de la position souhaitée, il se met dans sa configuration de dessin. Cela lui donne une position horizontale : son axe $\overrightarrow{Z}$ est normal au support de dessin. L'origine du repère de l'hexapode est alors à une hauteur H du sol. Selon cette valeur H, la zone de dessin varie ; or nous voulons maximiser son aire.

La fonction de calcul des angles (voir section VI-B) prend en entrée un point (x, y, z) ainsi qu'un des bras du robot, et détermine la configuration (q_1, q_2, q_3) correspondante. En comparant avec les angles minimaux et maximaux possibles pour chaque bras, on alors une fonction qui retourne un booléen : **True** si le point P est accessible par le bras k, **False** sinon. Cela détermine le champ d'action du bras, lorsque l'hexapode est au repos à une hauteur donnée.

Par exemple, on donne Fig. 14 le résultat de l'exécution de ce programme pour le bras avant gauche, à différents Z, pour les points (x, y) tels que $x, y \in [-400; 400]^2$, avec une résolution de 5mm.

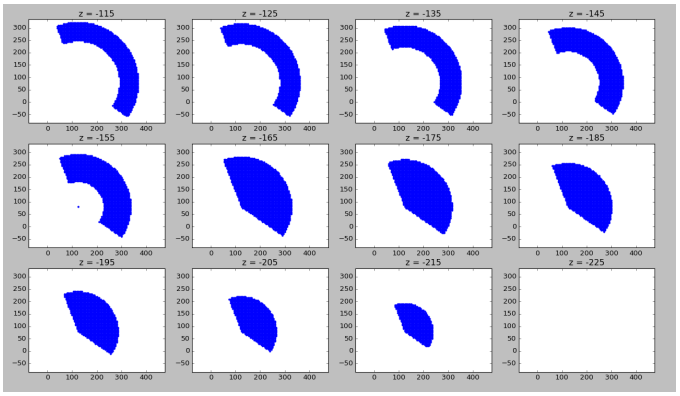


Fig. 14: Zone accessible par l'extrémité de la patte, pour différentes valeurs de Z

On obtient des allures de champ d'action qui correspondent avec l'intuition que l'on s'en fait, et pour maximiser la surface accessible, on pourrait sélectionner $H = -170mm$. Cependant il faut également prendre en compte l'intégration de l'outil de dessin, auquel on associe une translation $P = \begin{bmatrix} Tr_x & Tr_y & 0 \end{bmatrix}^T$ dans le repère du bout de la patte. En calculant l'aire maximale obtenue pour différentes valeurs de Tr_x et Tr_y, on obtient le résultat Fig. 15. Une claire augmentation de la zone accessible quand on augmente Tr_x, en revanche Tr_y n'a pas beaucoup d'influence dessus.

On a donc opté pour un feutre dont la pointe se trouve en $P = \begin{bmatrix} -50 & 150 & 0 \end{bmatrix}^T$. On montre Fig. 16 les zone accessible correspondantes, à différents Z, si on considère les 6 pattes simultanément. Une position de dessin avec $H = -165mm$ est une position optimale.

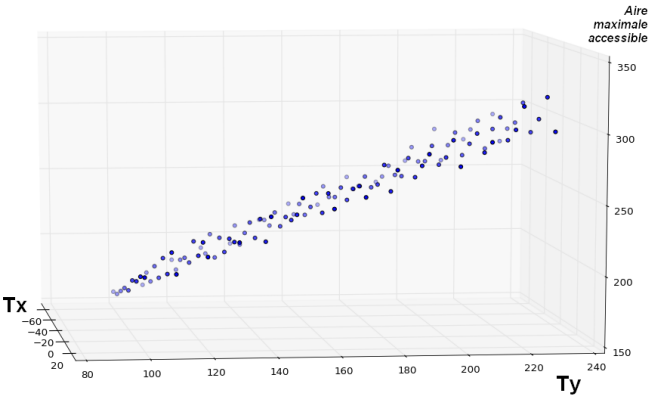


Fig. 15: Aire maximale obtenue, pour différentes positions Tr_x, Tr_y de l'extrémité de l'outil de dessin

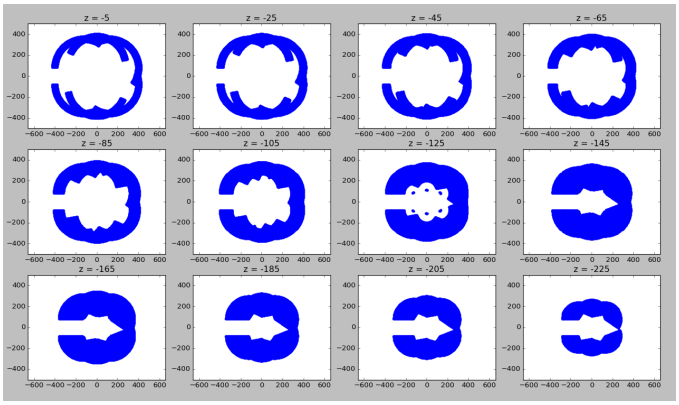


Fig. 16: Zone accessible par l'ensemble des six pattes pour différentes valeurs de Z

D. Changement de repère

L'intersection de ce masque avec les trajectoires à dessiner, effectuée à la position estimée du robot, donne les trajectoires directement réalisables par le robot. Les matrices de rotation et translation données par les caméras (section V-C) permettent d'exprimer les coordonnées des points dans le repère du robot, et on calcule les consignes moteurs à l'aide du modèle géométrique inverse pour tous les points de la trajectoire, ainsi que les points d'interpolation situés entre eux. Le robot relève le feutre entre deux trajectoires distinctes, et enlève de la liste les points dessinés.

VII. CONCLUSION

Ce projet Hexapode fut un challenge complexe pour notre groupe. D'un côté, cela nous a permis de rentrer dans le vif de la robotique : les robots hexapodes sont parmi les plus complets, proposant mécanique, électronique, programmation, et planification. De l'autre car nous avons dû en permanence faire preuve d'une totale innovation, étant donné que ce type de robot n'est pas fait initialement pour le dessin.

Perspectives pour l'amélioration :

Par manque du temps, et en vue d'améliorer ce travail, nous proposons quelques idées pour d'éventuels travaux à venir :

- Fusion des données du modèle de déplacement du robot avec les données du système de vision, pour améliorer la localisation utilisant un filtre de Kalman
- Utilisation du middleware ROS, pour améliorer l'organisation/l'architecture software du robot, et ainsi pouvoir améliorer la communication entre les différents programmes, améliorer les programmes séparément, et ajouter de nouvelles fonctionnalités.
- Améliorer la modélisation géométrique et cinématique du robot pour pouvoir synthétiser une commande plus performante.
- Intégrer un réseau de neurones, lui permettant d'apprendre en autonomie à réaliser diverses tâches

REMERCIEMENTS

Merci à l'équipe pédagogique des Masters SPI, qui nous ont toujours fourni un lieu de travail sérieux mais convivial. Merci au groupe Projet Hexapode de l'an passé, pour nous avoir aiguillé dans la bonne direction en début de projet. Merci à nos tuteurs V.Padois et M.Boudaoud pour leurs avis éclairés. Merci aux autres groupes de projet intégratif, pour le soutien technique et parfois moral apporté durant les dernières semaines.

REFERENCES

[1] Andrew Conru. The robotic art competition. https ://robotart.org/.

[2] Brett Kennedy, Hrand Agazarian, Yang Cheng, Michael Garrett, Gregory Hickey, Terry Huntsberger, Lee Magnone, Colin Mahoney, Amy Meyer, and Jennifer Knight. Lemur : Legged excursion mechanical utility rover. *Autonomous Robots*, 11(3) :201–205, 2001.

[3] Uluc Saranli, Martin Buehler, and Daniel E Koditschek. Rhex : A simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20(7) :616–631, 2001.

[4] David Henderson. Hexapod for tracking and following. https ://github.com/davidhend/Hexapod/, 2015.

[5] Kevin M. Ochs. Golem. https ://github.com/KevinOchs/hexapod_ros, 2016.

[6] Kåre Halvorsen. Morphex. http ://zentasrobots.com/robot-projects/morphex-mkiii/, 2014.

[7] John Canny. A computational approach to edge detection. In *Readings in Computer Vision*, pages 184–203. Elsevier, 1987.

[8] AP Bessonov and NV Umnov. The analysis of gaits in six-legged vehicles according to their static stability. In *On Theory and Practice of Robots and Manipulators*, pages 1–10. Springer, 1974.

[9] Baptiste Charmette. *Localisation temps-réel d'un robot par vision monoculaire et fusion multicapteurs*. PhD thesis, Université Blaise Pascal-Clermont-Ferrand II, 2012.

[10] Stephen Se, David G Lowe, and James J Little. Vision-based global localization and mapping for mobile robots. *IEEE Transactions on robotics*, 21(3), 2005.

[11] Andrej Babinec, Ladislav Jurišica, Peter Hubinskỳ, and František Duchoň. Visual localization of mobile robot using artificial markers. *Procedia Engineering*, 96 :1–9, 2014.

[12] Rafael Munoz-Salinas. Aruco : a minimal library for augmented reality applications based on opencv. *Universidad de Córdoba*, 2012.

[13] Detection of aruco markers. https ://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_d 2015.

[14] YI Abdel-Aziz. Direct linear transformation from comparator coordinates in close-range photogrammetry. In *Proceedings American society of photogrammetry symposium on close-range photogrammetry. Falls Church (VA). American Society of Photogrammetry Symposium on Close-Range Photogrammetry., 1971*, pages 1–19, 1971.