Travaux pratiques
ESIR 3 – Vrob2

TP d'asservissement visuel
Janvier/février 2025

Documents à rendre

- Le compte-rendu doit répondre aux questions posées dans le sujet et être illustré avec des images issues des scripts que vous avez complétés. Attachez une importance particulière à décrire ce que vous avez fait, expliquer pourquoi et comment vous l'avez fait, décrire les résultats obtenus, les analyser, faire le lien entre ces résultats et le cours.
- Le code Python complété doit contenir des commentaires qui doivent aider à comprendre l'algorithme développé.

TP d'asservissement visuel

1 Rappels et compléments du cours

Les techniques d'asservissement visuel consistent à élaborer des lois de commande sur le mouvement d'une caméra afin de réaliser des tâches de positionnement ou de suivi d'objets mobiles. Les informations visuelles utilisées dans la boucle de commande, notées x, doivent atteindre une valeur désirée x^*. Le problème de la commande se ramène alors à générer les mouvements de la caméra permettant de minimiser $(x - x^*)$ dans l'image. Une telle loi de commande en boucle fermée est donnée par :

$$\mathrm{v_c} = -\lambda \mathbf{L}_\mathrm{x}^{+}(\mathrm{x} - \mathrm{x}^*)$$

où :

- $\mathbf{v}_c = \left(v_x, v_y, v_z, \omega_x, \omega_y, \omega_z\right)$ est le torseur cinématique de la caméra ;
- λ est le gain qui règle la rapidité de convergence de la loi de commande ;
- L_x est la matrice d'interaction associée à x (définie par $x = L_x v_c$ et de dimension k × n où k est le nombre d'informations visuelles utilisées et n le nombre de degrés de liberté de la caméra) ;
- $\mathrm{L}_\mathrm{x}^{+}$ est la pseudo inverse de L_x (de dimension n × k) ;

On considèrera par la suite que la caméra a six degrés de liberté (n = 6).

2 Introduction

L'objectif de ce TP est de simuler plusieurs tâches d'asservissement visuel. Ce TP sera réalisé en Python.

3 Réalisation d'un démonstrateur d'asservissement visuel

L'objectif est ici de réaliser un démonstrateur permettant de simuler des tâches d'asservissement visuel. Votre travail sera de comparer et de commenter le comportement de lois de commande 2D et de lois de commande 3D. Vous pourrez vous aidez de la trame de code qui vous est fournie et la compléter.

3.1 Changement de repère et projection

Étant donné les coordonnées d'un point 3D ${}^{\mathrm{w}}\mathbf{X}$ exprimées dans le repère de la scène, vous écrirez une fonction permettant de calculer la position de ce point dans l'image vue depuis une position de la caméra donnée par la matrice homogène ${}^{\mathrm{c}}\mathrm{T}_\mathrm{w}$ (qui désigne la position de la caméra par rapport à la scène F_w).
Les coordonnées X d'un point 3D sont représentées par des vecteurs à trois composantes $\mathbf{X} = (\mathrm{X}, \mathrm{Y}, \mathrm{Z})^{\mathrm{T}}$.

Question 1 : Dans le fichier **Point.py** complétez la fonction **changeFrame(...)** qui permet d'obtenir les coordonnées d'un point 3D exprimées dans le repère F_a en fonction de ses coordonnées exprimées dans le repère F_b et la matrice de transformation rigide ${}^{\mathrm{a}}\mathbf{T}_\mathrm{b}$. Pour mémoire ${}^{\mathrm{a}}\mathbf{T}_\mathrm{b}$ est définie par :

$${}^{a}\mathbf{T}_b = \begin{pmatrix} {}^{a}\mathbf{R}_b & {}^{a}\mathbf{t}_b \\ 0 & 1 \end{pmatrix}$$

Question 2 : Les coordonnées x d'un point 2D sont représentées par des vecteurs à deux composantes $x = (x, y)^T$. Dans le fichier **Point.py** complétez la fonction **project(...)** qui permet d'obtenir les coordonnées d'un point 2D par projection perspective à partir des coordonnées d'un point 3D exprimées dans le repère de la caméra F_c.

3.2 Asservissement visuel sur un point 2D

Vous compléterez le code de **tp4-ibvs-one-point.py** qui vise à mettre en œuvre un asservissement visuel basé image sur un point 2D. Les fonctions permettant d'implémenter la loi de commande sont entièrement à écrire. Celles permettant l'affichage des courbes est donnée.

Conditions expérimentales :

1. L'objectif est de voir le point au centre de l'image soit $\mathbf{x}^* = (0,0)^{\mathrm{T}}$
2. Les coordonnées normalisées ${}^{\mathrm{w}}\mathbf{X}$ du point qui nous intéresse dans le repère monde F_w sont données par ${}^{\mathrm{w}}\mathbf{X} = (0.5, 0.2, -0.5,\ 1)^{\mathrm{T}}$
3. La position initiale de la caméra est la suivante :

$$ {}^{c}\mathbf{T}_{w} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

Question 3 : Donner la taille et la forme de la matrice d'interaction L_x associée à un point 2D. Dans le fichier **Point.py** compléter la fonction **interactionMatrix(...)** en conséquence.

Question 4 : Dans **tp4-ibvs-one-point.py**, mettre en œuvre la loi de commande permettant de déplacer la caméra afin d'observer le point à la position $\mathbf{x}^* = (0,0)^{\mathrm{T}}$.

Question 5 : Vous analyserez les résultats obtenus, en particulier l'erreur pour chaque primitive $|| \mathbf{x_i} - \mathbf{x_i}^* ||$, la norme globale de l'erreur $|| \mathbf{x} - \mathbf{x}^* ||$, la vitesse v de la caméra, la trajectoire de la position du point x dans l'image, la trajectoire de la caméra.

3.3 Asservissement visuel sur quatre points 2D

Vous compléterez le code de **tp4-ibvs-four-points.py** qui vise à mettre en œuvre un asservissement visuel basé image sur quatre points 2D.

Conditions expérimentales :

1. Quatre points sont positionnés aux sommets d'un carré aux positions suivantes dans le repère w : ${}^{w}\mathbf{X} = (-L, L, 0)^T, (L, L, 0)^T, (L, -L, 0)^T, (-L, -L, 0)^T$
   Nous choisirons L = 0.1m.
2. La position désirée de la caméra est la suivante :

$$ {}^{c^*}\mathbf{T}_{w} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} $$

3. Les positions initiales de la caméra pour lesquelles vous mènerez les tests sont données par la matrice de rotation et le vecteur de translation suivants :
   - Cas 1: c_R_w = RotationMatrix.buildFromThetaUVector([0, 0, 0]) et c_t_w = [0, 0, 1.3]
   - Cas 2: c_R_w = RotationMatrix.buildFromThetaUVector([Math.deg2rad(10), Math.deg2rad(20), Math.deg2rad(30)]) et c_t_w = [-0.2, -0.1, 1.3]

   Nota : Utilisez l'option « --initial-position » pour changer de position initiale.

Question 6 : Dans **tp4-ibvs-four-points.py i**mplémentez la loi de commande pour réaliser cette tâche. Pour chaque position initiale, vous analyserez les résultats obtenus, en particulier l'erreur pour chaque primitive $|| \mathbf{x}_i - \mathbf{x}_i^* ||$, la norme globale de l'erreur $|| \mathbf{x} - \mathbf{x}^* ||$, la vitesse v de la caméra, la trajectoire de la position du point x dans l'image, la trajectoire de la caméra.

Question 7 : Comparez plusieurs lois de commande :

- avec la matrice d'interaction évaluée à la position courante $\mathbf{L}_{\mathbf{x}|\mathbf{x}=\mathbf{x}(t)}$

- avec la matrice d'interaction évaluée à la position désirée $\mathbf{L}_{\mathbf{x}|\mathbf{x}=\mathbf{x}^*}$

Modifiez le script python pour gérer ces deux cas.

Nota : Utilisez l'option « --interaction » pour changer le type de matrice d'interaction.

Question 8 : Commenter et comparez les résultats obtenus en changeant le type de matrice d'interaction.

Question 9 : Vous ferrez aussi les tests pour les deux positions suivantes de la caméra correspondants à une simple rotation de 90 degrés (sur l'axe z) par rapport à la position désirée. Que constatez-vous ? Même question avec un rotation de 180 degrés. Commentez.

- Cas 3: c_R_w = RotationMatrix.buildFromThetaUVector([0, 0, Math.deg2rad(90)]) et c_t_w = [0, 0, 1]
- Cas 4: c_R_w = RotationMatrix.buildFromThetaUVector([0, 0, Math.deg2rad(180)]) et c_t_w = [0, 0, 1]

Nota : Utilisez l'option « --initial-position » pour changer de position initiale.

3.4 Asservissement visuel 3D

Vous compléterez le code de **tp4-pbvs-four-points.py** qui vise à mettre en œuvre un asservissement visuel basé sur la pose de la caméra. L'objectif est donc de mettre en œuvre une loi de commande 3D. Pour mémoire, celle-ci vise à minimiser l'erreur entre la position courante ${}^{\mathrm{c}}T_{\mathrm{w}}$ et la position désirée ${}^{\mathrm{c}^*}T_{\mathrm{w}}$ de la caméra.

Question 10 : Comment calculer l'erreur ${}^{c^*}T_{\mathrm{c}}$ et le vecteur d'erreur associé ? Quelle est la loi de contrôle ? Vous implémenterez cette loi de commande dans **tp4-pbvs-four-points.py**.

Question 11 : Implémentez la loi de commande pour réaliser cette tâche. Nous reprendrons les mêmes conditions expérimentales que dans la section précédente et vous testerez donc les quatre positions initiales précédentes Pour chaque position initiale, vous analyserez les résultats obtenus, en particulier l'erreur pour chaque primitive $||\ \mathbf{x}_i - \mathbf{x}_i{}^*\ ||$, la norme globale de l'erreur $||\ \mathbf{x} - \mathbf{x}^*\ ||$, la vitesse v de la caméra, la trajectoire de la position du point x dans l'image, la trajectoire de la caméra.

Nota : Utilisez l'option « --initial-position » pour changer de position initiale.

3.5 Poursuite d'une cible mobile

Vous reprendrez le code de **tp4-ibvs-four-points.py** qui vise à mettre en œuvre un asservissement visuel basé image sur quatre points 2D et utiliserez l'option « --move-target » qui permet de mettre en mouvement l'objet avec un mouvement de translation dans le repère F_w.

Question 12 : Selon quel axe et à quelle vitesse se déplace l'objet ?

Question 13 : Lancez une simulation en utilisant l'option « --move-target ». Que constatez-vous ?

Question 14 : Dans **tp4-ibvs-four-points.py** proposez une méthode pour résoudre les éventuels problèmes observés en modifiant la loi de commande pour prendre en compte le mouvement de l'objet. Lancez une simulation. Que constatez-vous ?