Dugne Quentin - Glédel Clément

Asservissement visuel

L'objectif de ce TP est de simuler plusieurs tâches d'asservissement visuel par le biais de loi de commande 2D et de lois de commande 3D.

Nous commencerons simplement par faire bouger une caméra afin d'observer un point à un endroit précis de l'image (centre). Ce point sera exprimé par les coordonnées monde suivante : ${}^{w}\mathbf{X} = (0.5, 0.2, -0.5)^{\top}$ Il nous faut donc une fonction capable de convertir les coordonnées monde d'un point en coordonnées image.

Pour rappel ${}^{b}X_a = {}^{a}T_b * {}^{b}X$: où T représente la transformation rigide entre a et b.

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

A cela nous avons besoin d'une fonction permettant d'obtenir les coordonnées 2D des points en projetant les points 3D exprimées dans le repère de la caméra sur un plan 2D. Pour cela il suffit de diviser les coordonnées par la 3ème composante Z.

Nous pouvons maintenant calculer la matrice d'interaction après avoir définis les positions initiales des points dans le repère caméra en utilisant comme matrice de transformation entre le repère monde et caméra la matrice homogène suivante :

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

Les dimensions de matrice d'interaction associé à x dépend du nombre de points que l'on suit. Dans notre cas nous suivons un point, nous aurons donc une matrice d'interaction de dimension (2,6).

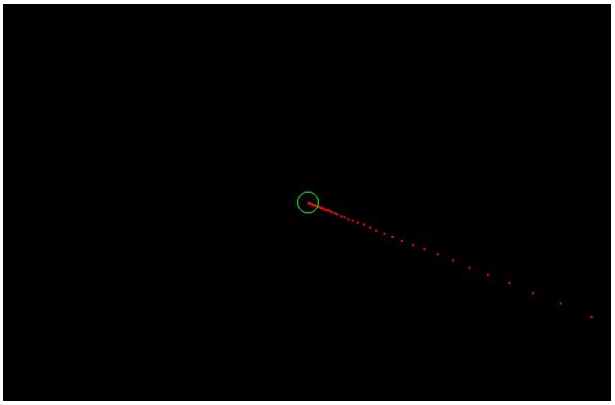
Elle s'écrit sous la forme suivante où cX représente le vecteur de la position courante en 3D et dX le vecteur de la position d'un point 2D (qui peut être désirée ou courante en fonction de la loi de commande à utiliser). Dans notre premier cas nous prenons dX comme étant le point 2D désiré :

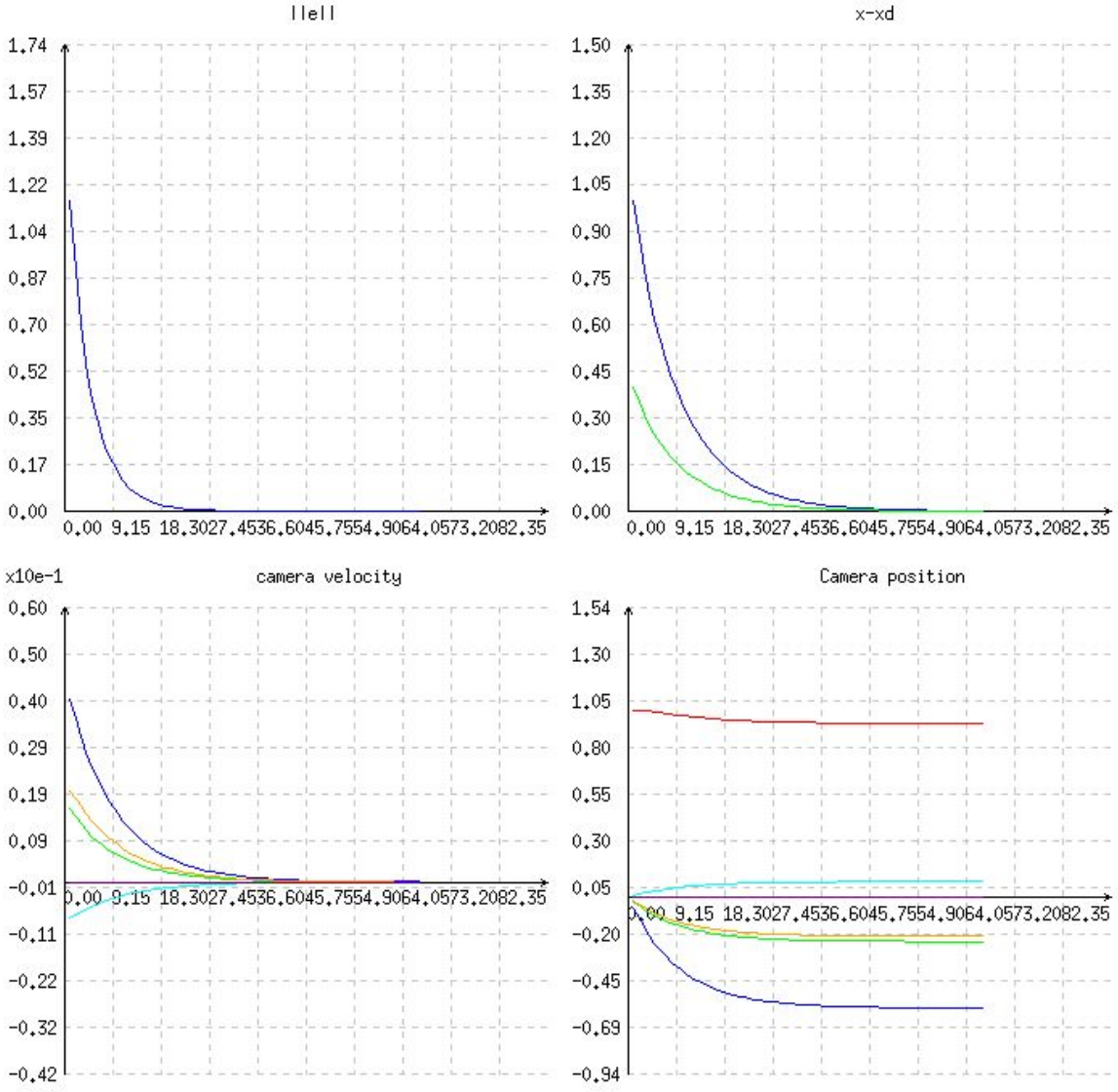
$$ L_x = \begin{matrix} -1/cX_z & 0 & dX_x/cX_z & dX_x * dX_y & -1 - dX_x^2 & dX_y \\ 0 & -1/cX_z & dX_y/cX_z & 1 + dX_y^2 & -dX_x * dX_y & -dX_x \end{matrix} $$

Après avoir calculé l'erreur entre la position désirée et la position courante dans l'image nous pouvons appliquer la loi de commande décrite par cette formule:

$$ \mathbf{v}_c = -\lambda \, \mathbf{L}_{\mathbf{x}}^{+} (\mathbf{x} - \mathbf{x}^{*}) $$

Nous obtenons les résultats suivant :





Les graphiques obtenues permettent de voir l'évolution des paramètres que l'on fait varier afin de se rapprocher de la position désirée. Les 2 premiers graphiques représentent l'erreur en valeur absolue ainsi que l'erreur suivant chaque axe. Nous pouvons dès à présent voir que ces graphiques sont cohérents avec l'image du déplacement du point dans le repère de la caméra. En effet, l'erreur suivant y démarre plus faible que l'erreur suivant x et décroissent jusqu'à atteindre 0, soit la position désirée.

Les graphiques inférieurs représentent l'évolution des 6 composantes du vecteur vitesse v de la loi de commande ainsi que la variation de la position de caméra suivant ses 6 degrés de libertés. Encore

une fois les valeurs sont cohérentes sachant que nous souhaitons que les composantes de la vitesse convergent toutes vers 0 et que les DOF de la caméra se stabilisent.

Cette fois ci nous ne travaillerons pas avec un point mais avec quatre ayant pour coordonnées monde :

$$((-L, L, 0), (L, L, 0), (L, -L, 0), (-L, -L, 0))$$

avec L = 0.3m

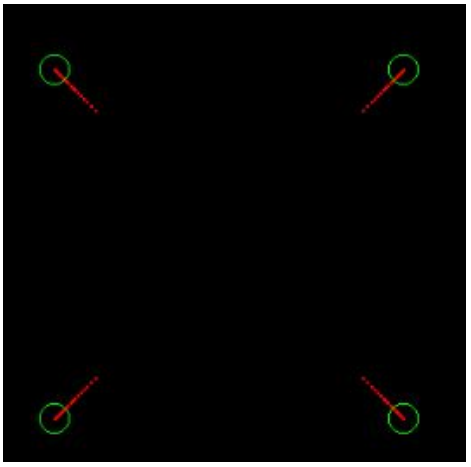
Nous réitérons la même opération qu'auparavant excepté que cette fois ci comme nous avons quatre points, nous aurons donc une matrice d'interaction de dimension (8,6).

Dans les exemples suivants nous effectuerons nos test selon 4 positions initiales de caméra. Chaque test comportera **2 cas** :

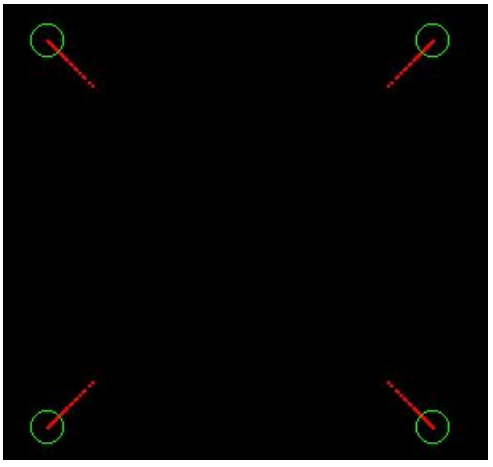
- Dans le premier cas, nous appliquerons la loi de commande avec la matrice d'interaction évaluée à la **position désirée**.
- Tandis que dans le deuxième cas, nous évaluerons la matrice d'interaction à la **position courante**.

Ainsi si l'on fixe la position initiale de la caméra selon les configurations suivantes, nous obtenons les résultats suivants :

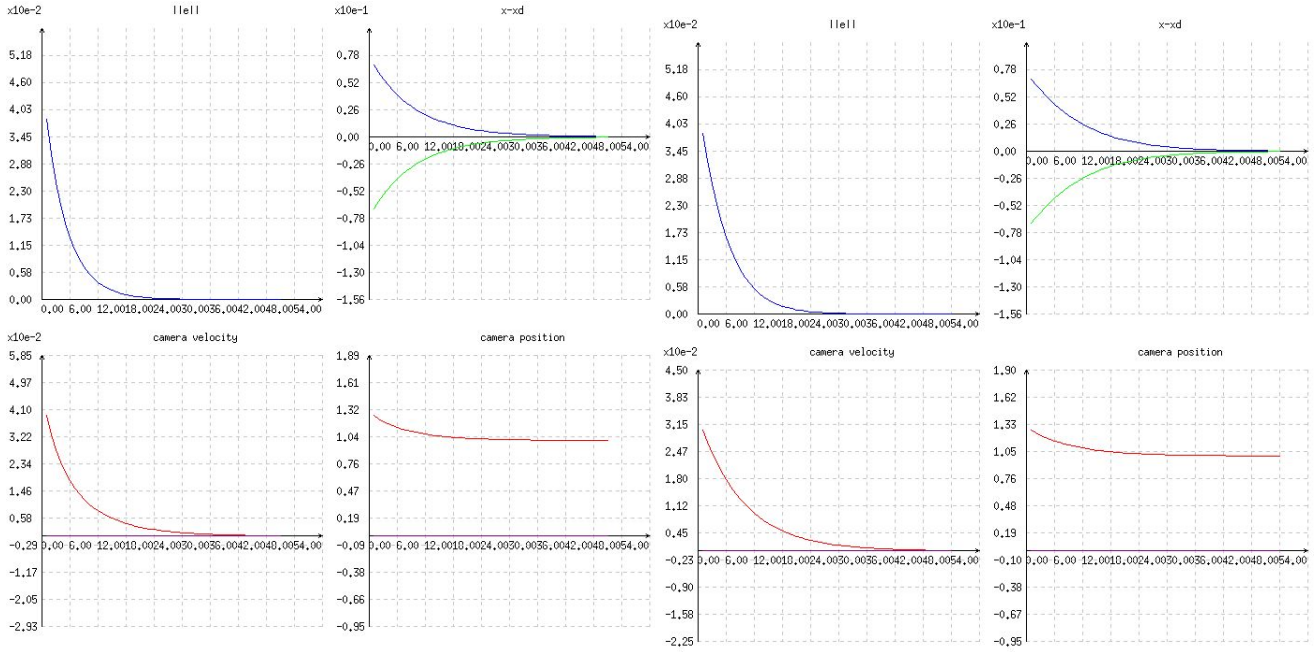
Première position de caméra : cTw (0, 0, 1.3, 0, 0, 0)



Cas 1

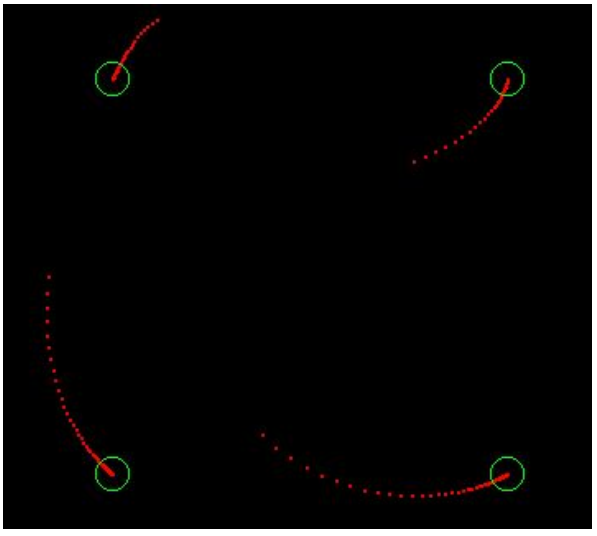


Cas 2

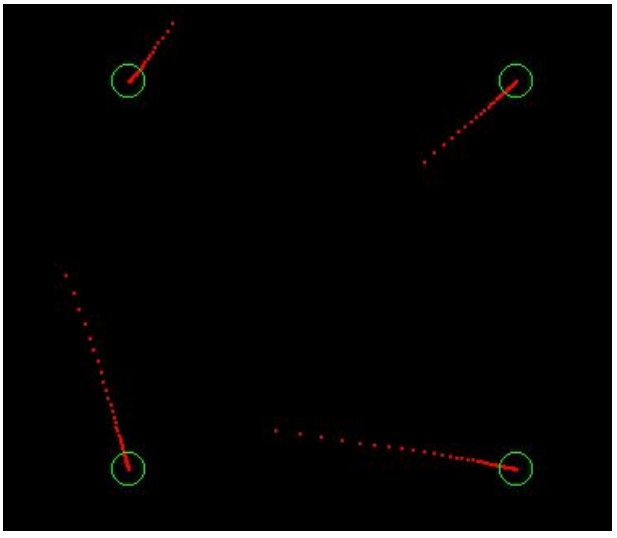


L'objectif de cette partie est d'observer les différences entre les deux lois de commande. Néanmoins, dans certain cas comme celui ci, où la position initiale de la caméra est similaire à un zoom, le résultat obtenu par les deux différentes lois de commande seront les mêmes. La caméra effectue les mêmes déplacement dans les deux cas et par conséquent il en est de même des points dans les deux images. On notera par la même occasion que la matrice d'interaction ne semble pas ou très peu influencer sur la vitesse de convergence de l'erreur comme le montre les courbes de la partie supérieure.

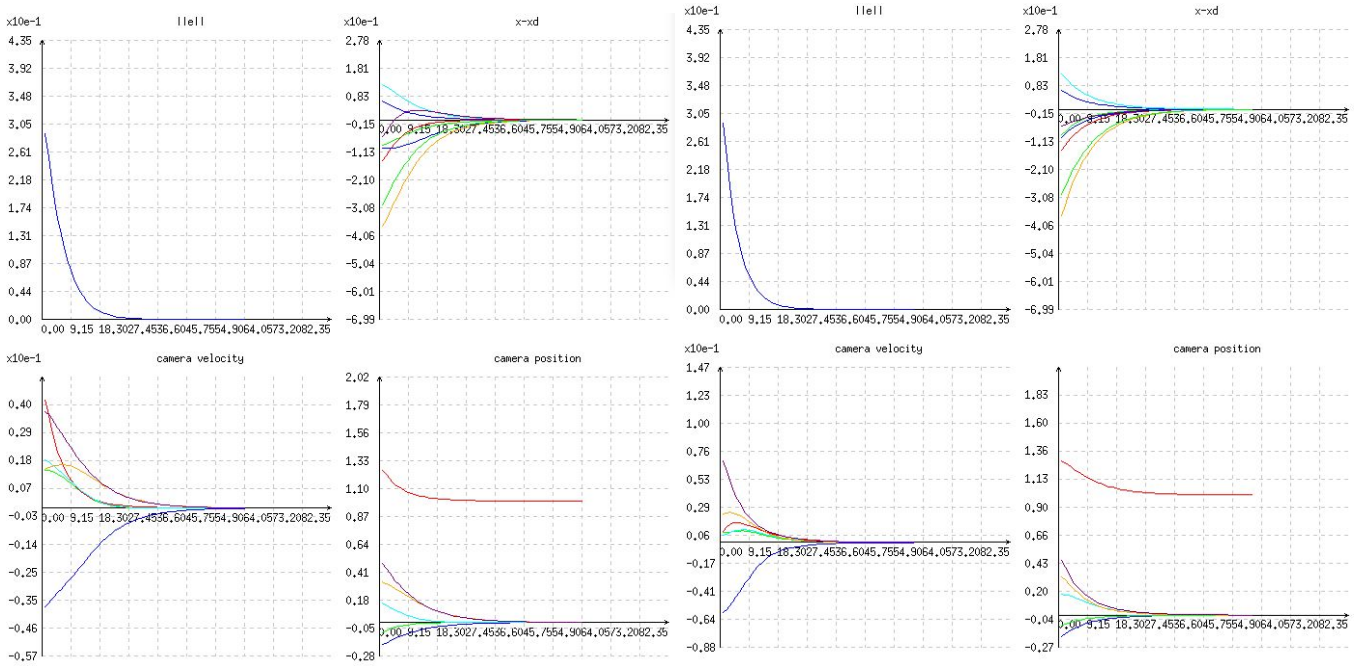
<u>Deuxième position de caméra :</u> cTw (-0.2, -0.1, 1.3, rad(10),rad(20), rad(30))



cas 1



cas 2

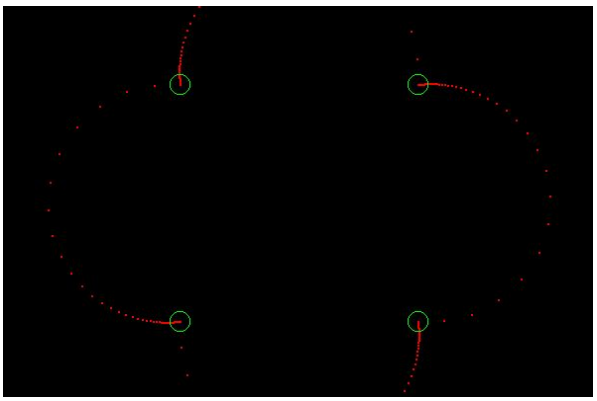


Cette fois-ci la position initiale est légèrement plus complexe puisqu'il ne suffit plus d'un simple zoom. La caméra a subi une légère rotation ainsi qu'une translation selon chacun de ses axes. La vitesse de convergence de l'erreur est relativement similaire d'un cas à l'autre cependant on observe cette fois ci des différences sur les autres courbes. En effet comme le montre l'image du champs visuel de la caméra, les trajectoires sont courbés dans l'une et droites dans l'autre. C'est le principe des 2 différentes matrices d'interactions.

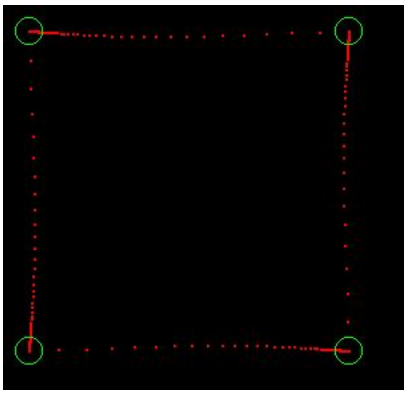
Lorsque l'on utilise la position courante *(cas 2)*, la matrice d'interaction est calculée à la position actuelle des points vue par la caméra et donc met en relation la position courante des points en 3D et en 2D, ce qui va entraîner un déplacement des points linéaire vers la position désirée du point de vue de la caméra. On peut remarquer que les trajectoires sont légèrement courbées mais tendent à être une droite. L'objectif lors de l'utilisation de cette loi de commande est d'obtenir une trajectoire de l'objet linéaire dans le champs visuel de la caméra.

A l'inverse dans l'autre cas *(cas 1)*, on calcul la matrice d'interaction avec la position désirée des points et donc met en relation la position courante des points en 3D et la position désirée en 2D, ce qui va entraîner un déplacement de la caméra un peu plus linéaire jusqu'à la position de la caméra souhaitée sans aucune contrainte de trajectoire linéaire des points. Dans ce cas, la trajectoire des points dans le plan de la caméra ne sera pas forcément linéaire. L'objectif lors de l'utilisation de cette loi de commande est d'éviter la contrainte de trajectoire linéaire des points qui peut déboucher sur des minimas locaux.

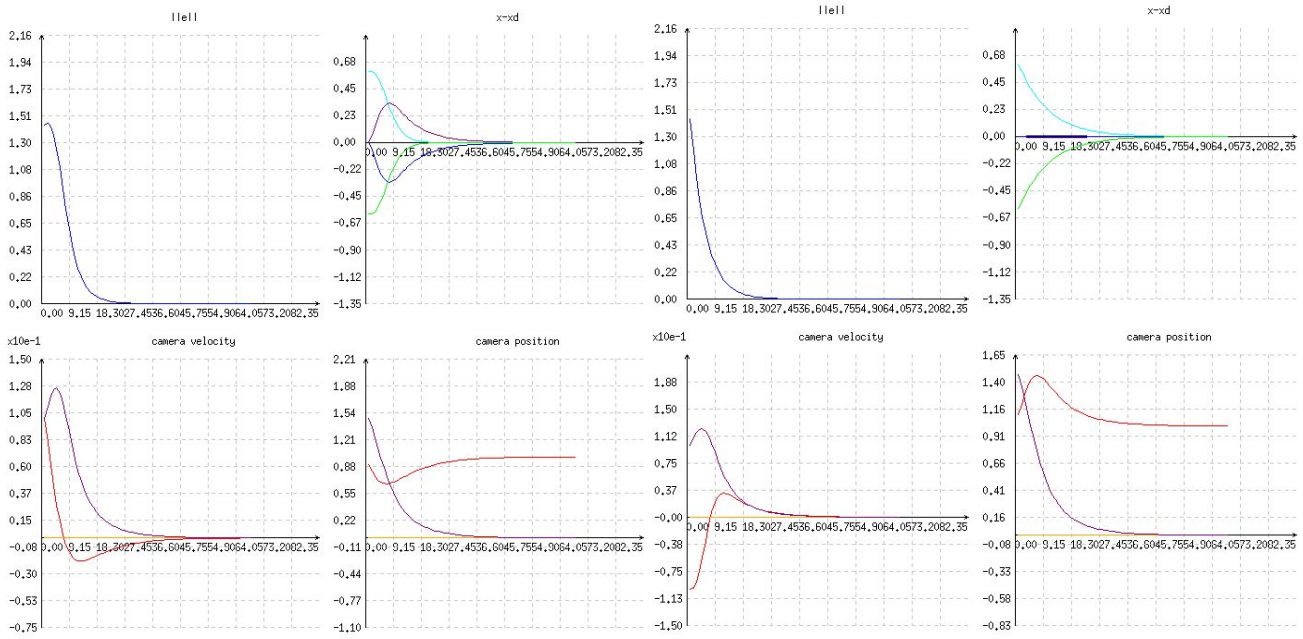
<u>Troisième position de caméra :</u> cTw (0, 0, 1, 0, 0, rad(90))



cas 1



cas 2

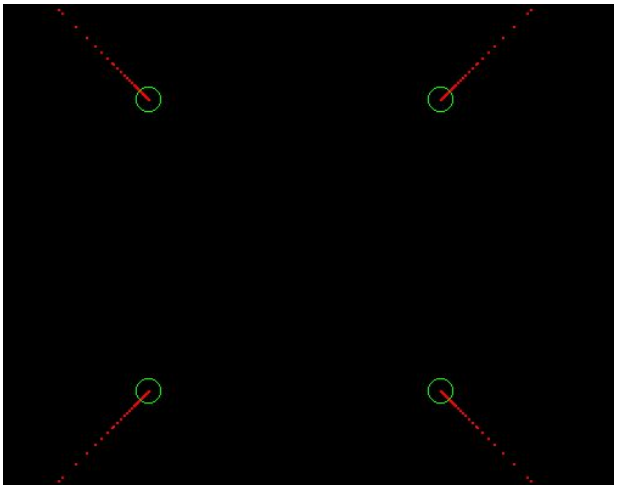


Cet exemple met en avant une caméra orientée de 90° selon l'axe des Z et légèrement zoomé par rapport aux positions désirées des points. Les remarques précédentes concernant la trajectoire des points dans l'image s'appliquent une nouvelle fois dans ce cas. Néanmoins on observe une légère différence dans la convergence de l'erreur car celle ci augmente légèrement lorsque l'on calcule la matrice d'interaction avec la position désirée.

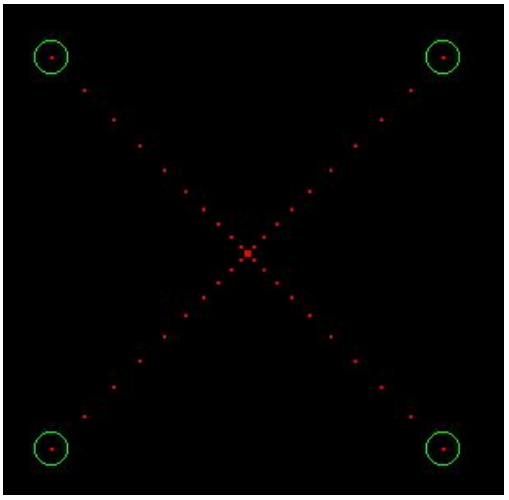
En effet, on peut remarquer que dans le cas 2, la trajectoire des points étant linéaire, on peut en déduire qu'afin de garder une trajectoire des points linéaire la caméra a dû effectuer une rotation tout en effectuant un dézoome plus important que nécessaire puis zoomer avant d'arriver à la position désirée *(visible sur le graphe camera velocity, rouge = translation Z et violet = rotation Z)*. Nous pouvons notamment l'observer sur le graphique x-xd, les coordonnées x et y de chaque point se rapproche de plus en plus sans augmenter l'erreur 2D entre la position courante et désirée au contraire du cas 1.

Dans le cas 1, ne cherchant pas à avoir une trajectoire des points linéaire, la caméra va effectuer la rotation et dézoomer puis zoomer légèrement jusqu'à la position désirée. Cela qui va entraîner une monté de l'erreur lorsqu'on effectue la rotation et que le zoom n'est pas assez prononcé (voir graphique x-xd).

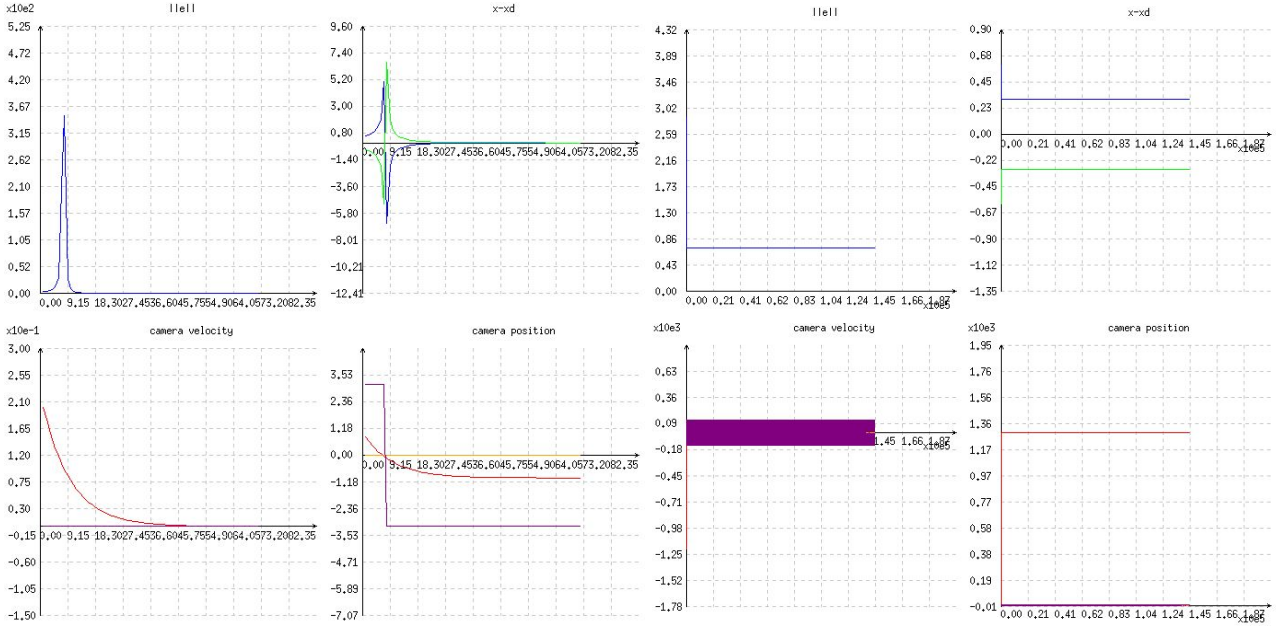
Quatrième position de caméra : cTw (0, 0, 1, 0, 0, rad(180))



cas 1



cas 2



Ce dernier cas est particulièrement intéressant car jusque là nous n'avons pas mis en avant les qualités de la matrice d'interaction selon les points désirés. L'intérêt de celle-ci est d'éviter ce qui est arrivé à son homologue à savoir les minimas locaux. En effet dans ce cas les points doivent se rendre à leur opposé par rapport au centre à cause de la rotation de 180°.

Dans le cas 2 qui utilise les positions courantes la matrice d'interaction permet d'obtenir un dézoome afin de faire avancer les points linéairement dans la bonne direction cependant un dézoome ne permet pas de compenser une rotation (ne permet pas de traverser le centre) et donc fini par se rendre dans un minima local.

Dans le cas 1, nous nous retrouvons pas coincé dans ce minima local car la loi de commande faite avec la position désirée permet de translater la caméra vers sa position désirée sans nécessité d'obtenir une trajectoire linéaire des points. Nous pouvons remarquer grâce aux graphiques que la caméra commence par effectuer une rotation de 180° (c'est pourquoi nous ne pouvons voir le mouvement de rotation sur l'image de la caméra) puis dézoome jusqu'à la position désirée.

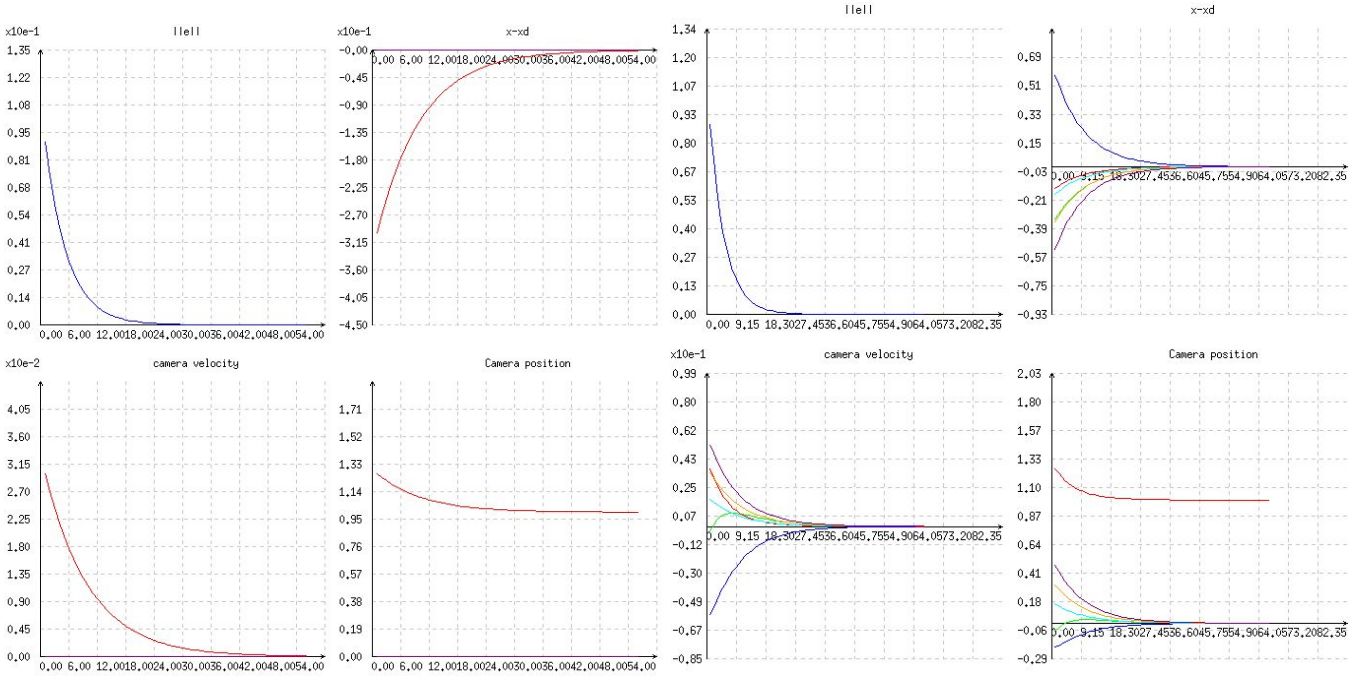
Asservissement visuel 3D

L'objectif est de minimiser l'erreur entre la position courante ${}^{c}\mathbf{T}_{w}$ et la position désirée ${}^{c^*}\mathbf{T}_{w}$ de la caméra. Pour cela nous calculons la matrice de transformation **cdTc** en multipliant la matrice homogène **cdTw** par l'inverse de la matrice homogène **cTw** puis nous en récupérons les valeurs de translations ainsi que celles de θu issues de la matrice de rotation afin de créer le vecteur d'erreur.

Le processus de création de la matrice homogène quant à elle varie légèrement de l'asservissement 2D puisque nous utilisons ces même valeurs extraites auparavant afin de calculer la formule suivante:

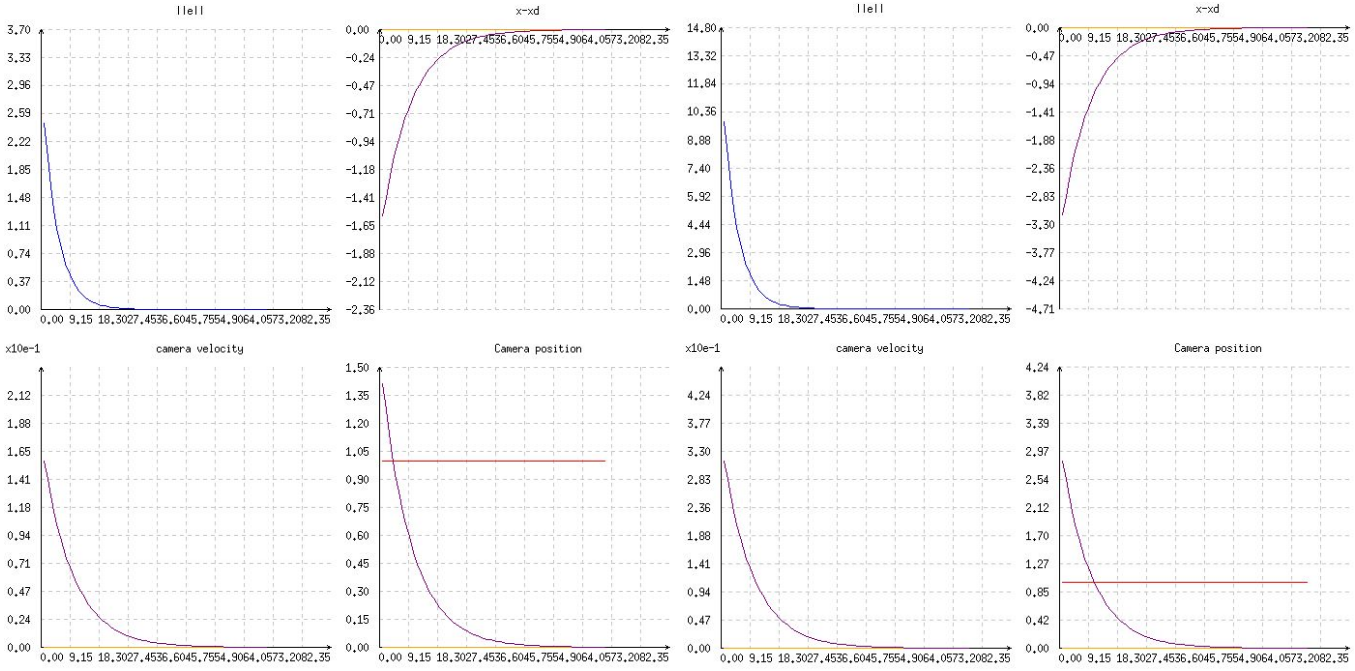
$$\mathbf{L}_{\omega}(\mathbf{u},\theta) = \mathbf{I}_3 - \frac{\theta}{2}[\mathbf{u}]_{\times} + \left(1 - \frac{\mathrm{sinc}(\theta)}{\mathrm{sinc}^2(\frac{\theta}{2})}\right)[\mathbf{u}]_{\times}^2$$

De cette matrice nous en déduisons la matrice d'interaction.



cTw (0, 0, 1.3, 0, 0, 0)

cTw (-0.2, -0.1, 1.3, rad(10),rad(20), rad(30))



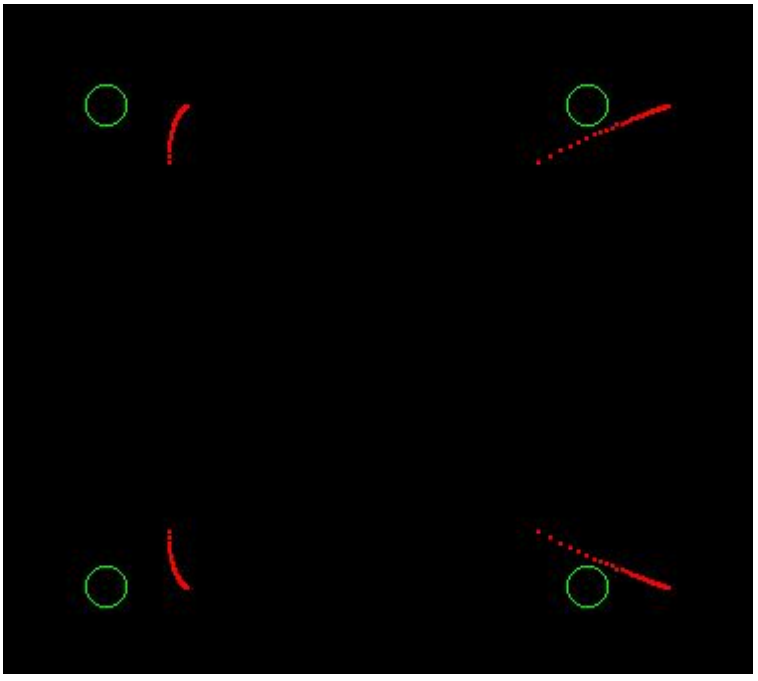
cTw (0, 0, 1, 0, 0, rad(90)) cTw (0, 0, 1, 0, 0, rad(180))

Notre implémentation de la loi de commande d'asservissement visuel 3D cherche à minimiser l'erreur faite avec le vecteur translation **cdtc** et le vecteur **thetau** récupéré avec dans la matrice homogène **cdTc.** Le grand intérêt de l'utilisation de ces caractéristiques 3D est d'obtenir une trajectoire de caméra en ligne droite dans l'espace 3D ainsi que de ne jamais tomber dans des minimas locaux. En effet, nous pouvons voir ci-dessus qu'aucun des graphiques représentant les erreurs (||e|| et x-xd) ne montre d'augmentation de l'erreur absolue (diminution constante), ce qui montre que la caméra effectue un mouvement en ligne droite sans mouvement autre.

Poursuite

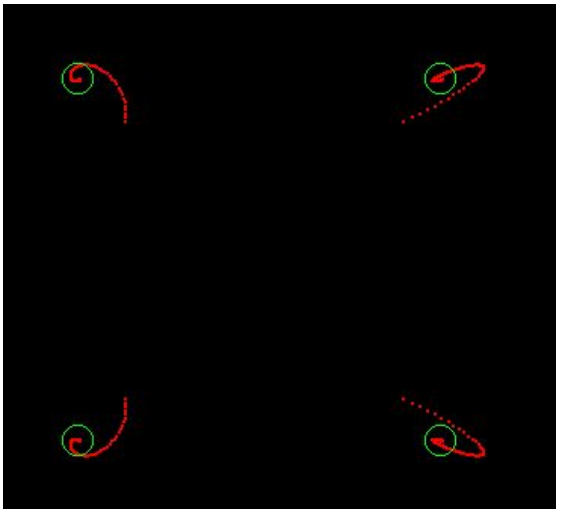
Dans cette partie nous reprenons notre asservissement visuel 2D et ajoutons un mouvement de translation de 1cm/s à nos 4 points suivant l'axe x. Le but de cette partie est d'implémenter une méthode permettant de réussir à atteindre l'objectif en mouvement.

Lors de l'ajout du déplacement un problème survient lorsque l'on souhaite atteindre les positions désirées. En effet, les points n'arrivent jamais à la position souhaitée et reste décalés derrière l'objectif, voir ci- dessous :

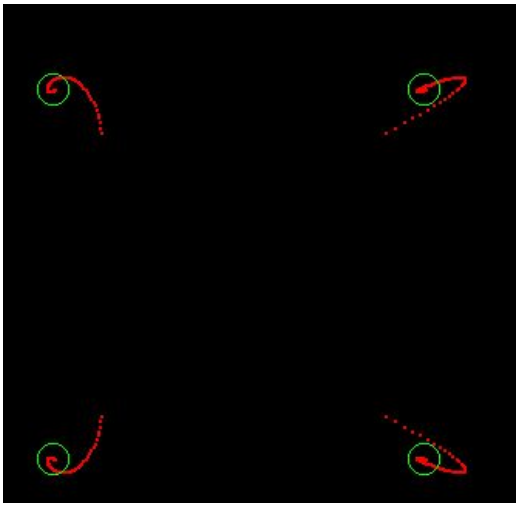


Afin de palier à ce problème nous ajoutons à l'erreur courante lors du calcul de la vitesse, une erreur égale à l'accumulation des erreurs obtenues jusque là à un facteur **nu** préfixé à 0.01 dans notre cas. Ainsi nous obtenons les résultats suivants :

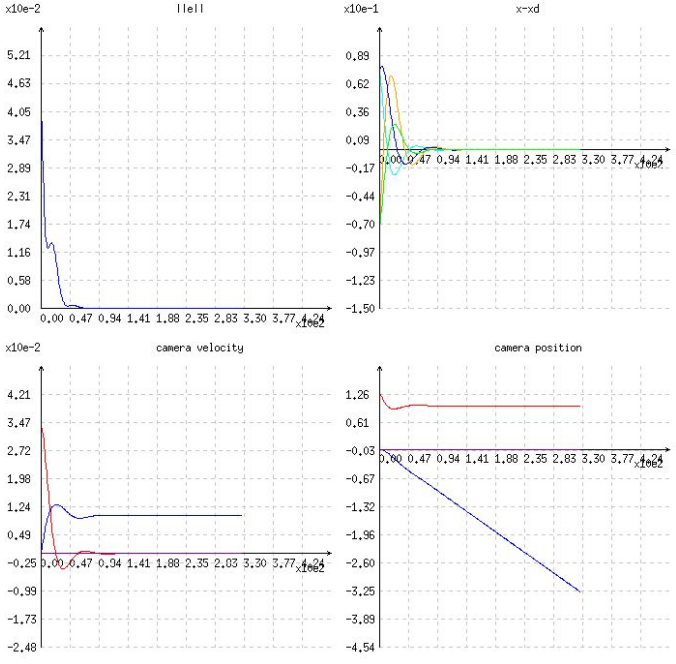
Première position de caméra : cTw (0,0,1.3, 0,0,0)

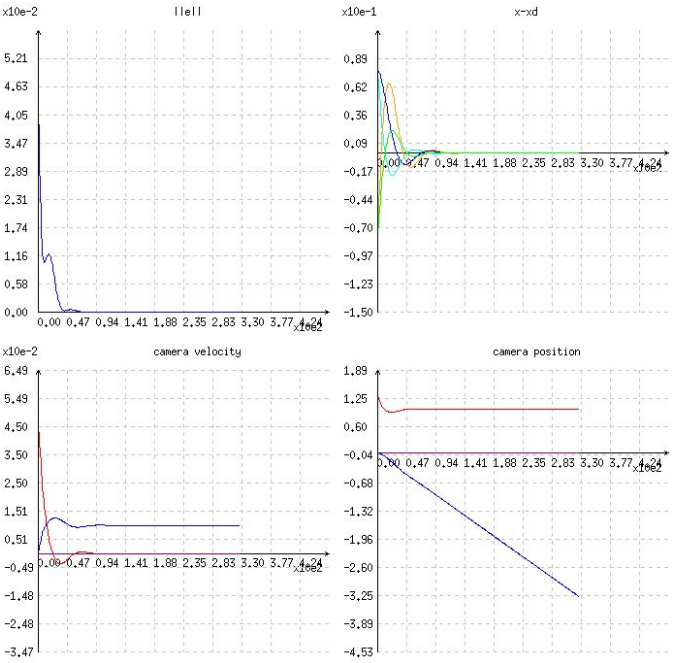


position désirée

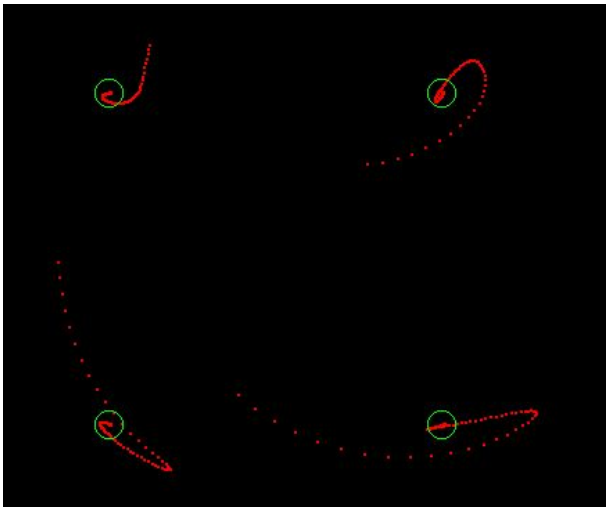


position courante

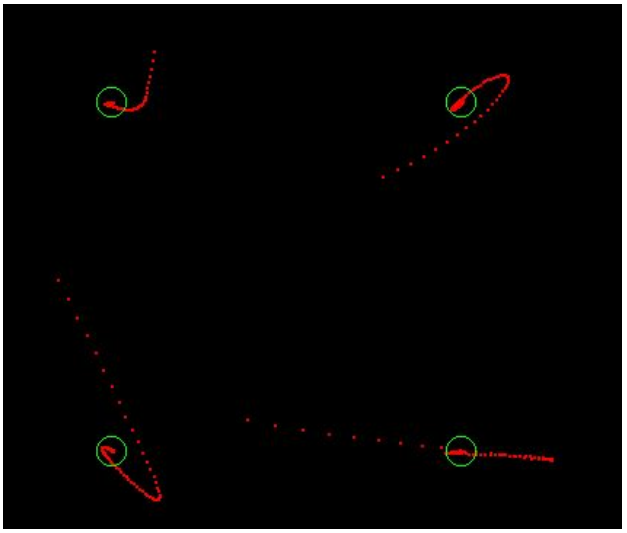




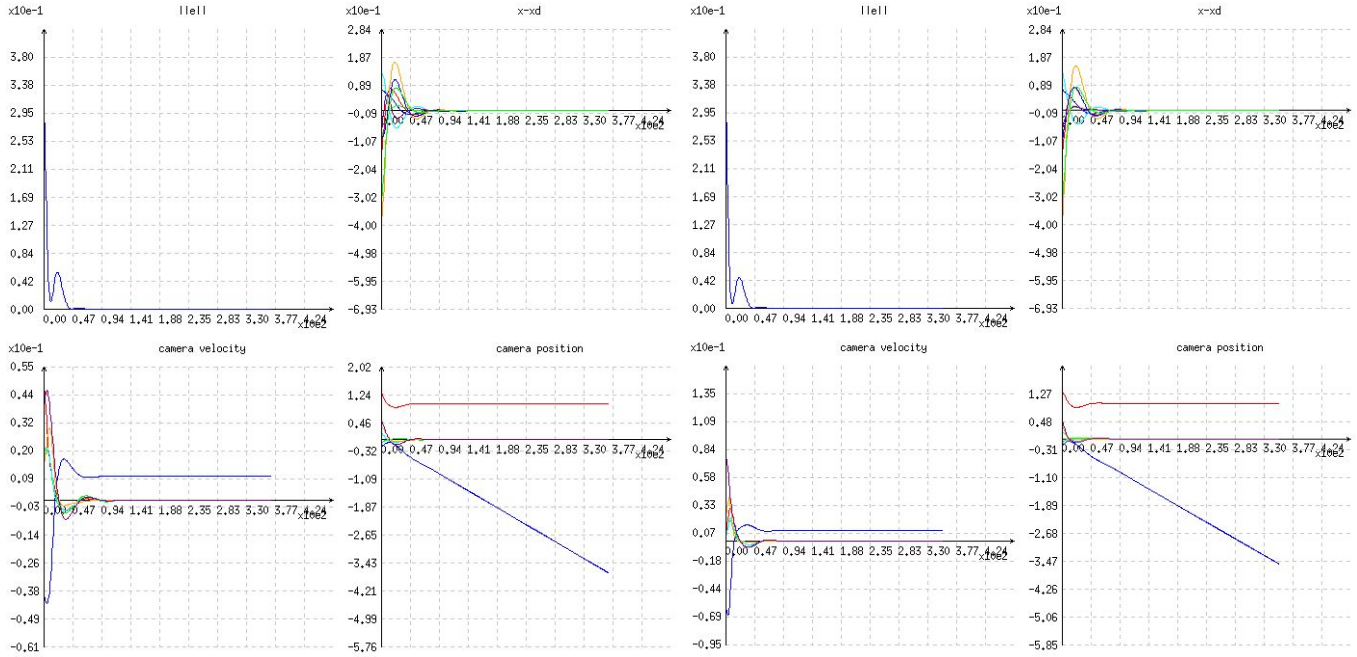
Deuxième position de caméra : cTw (-0.2, -0.1, 1.3, rad(10),rad(20), rad(30))



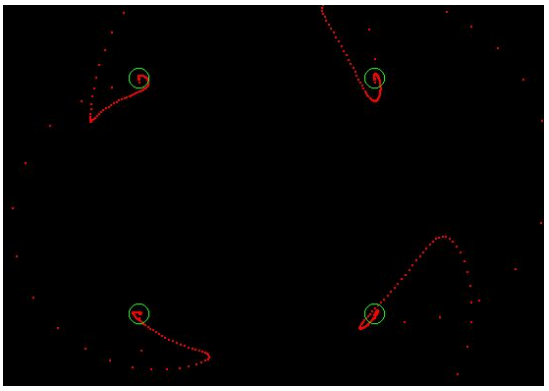
position désirée



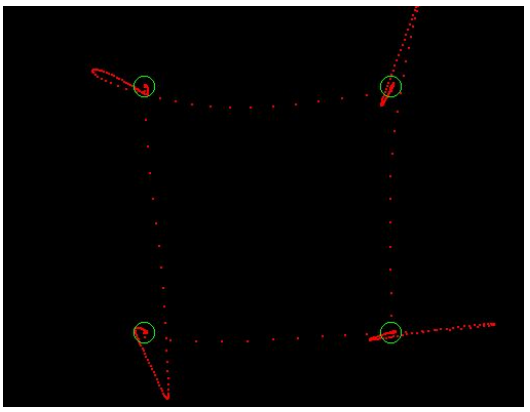
position courante



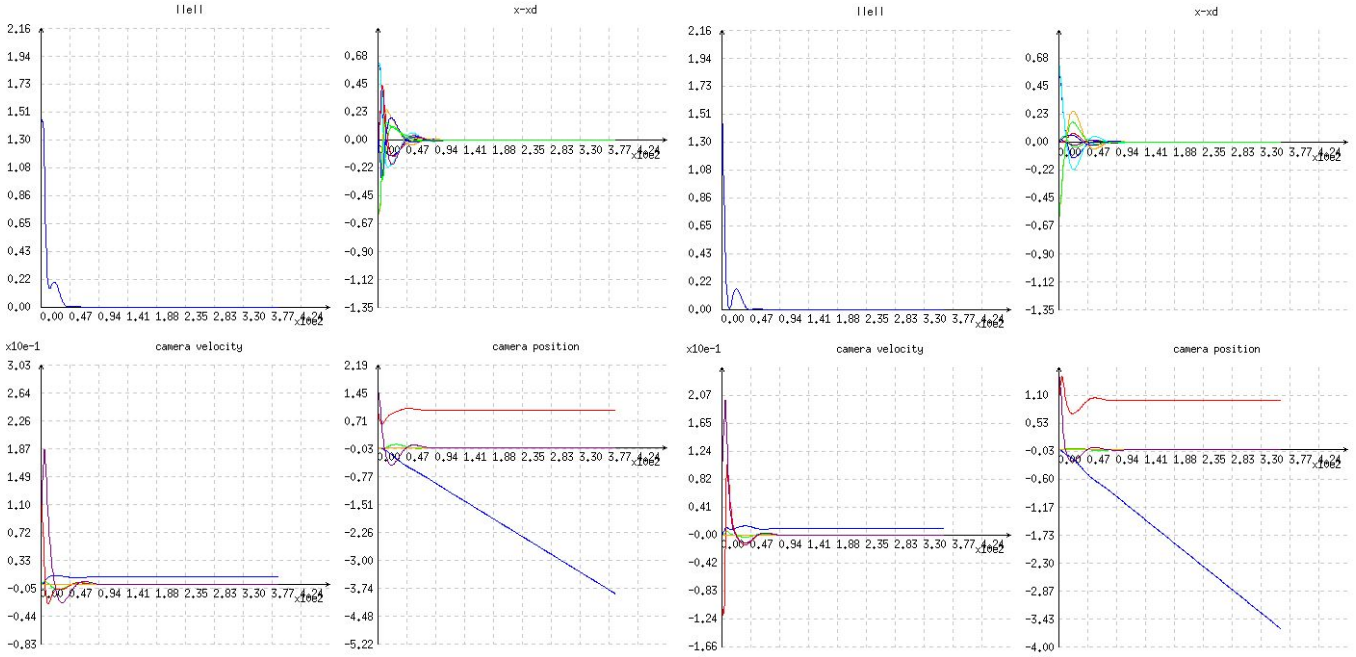
Troisième position de caméra : cTw (0, 0, 1, 0, 0, rad(90))



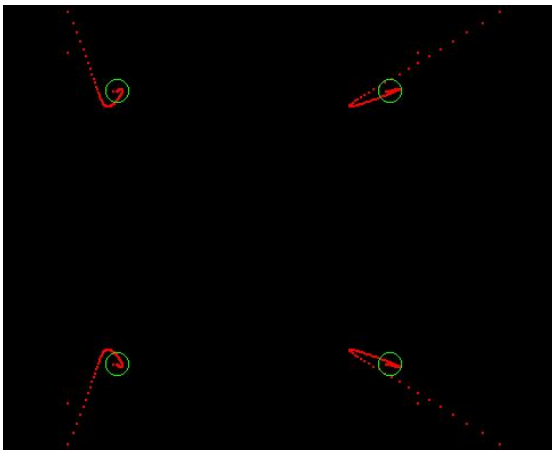
position désirée



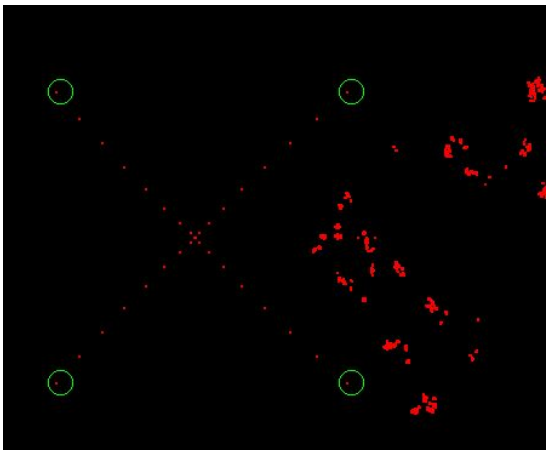
position courante



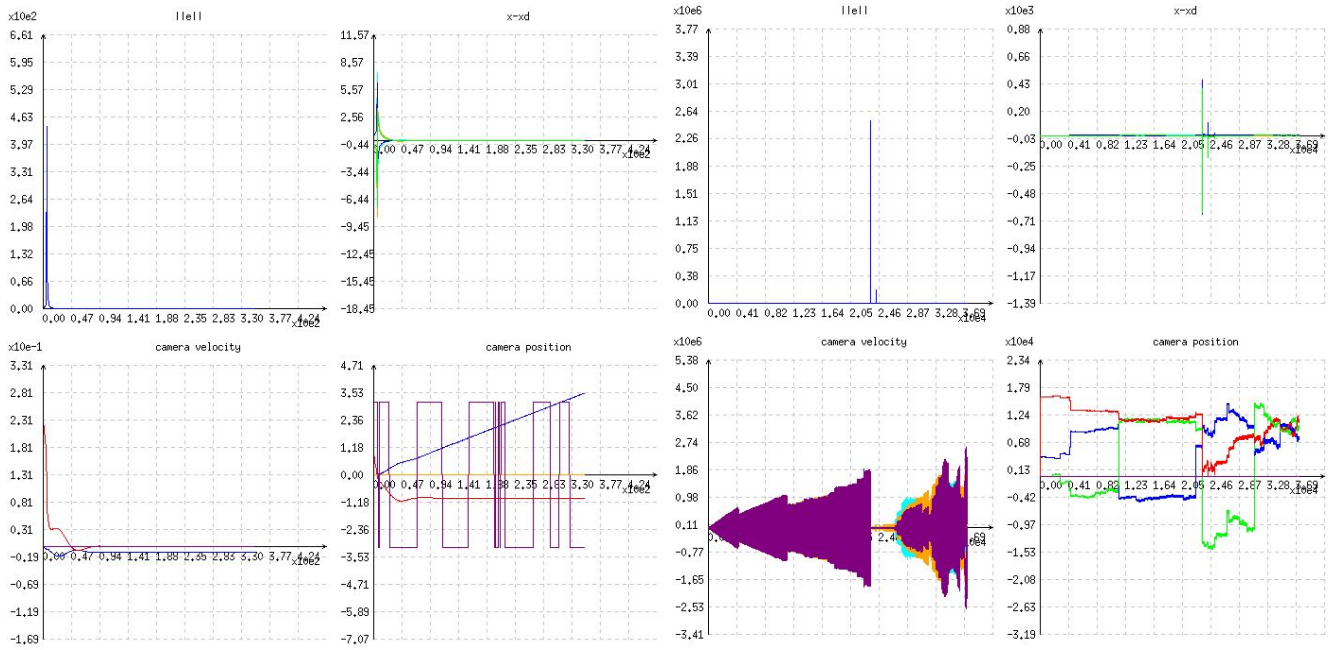
Quatrième position de caméra : cTw (0, 0, 1, 0, 0, rad(180))



position désirée



position courante



Tout d'abord nous pouvons remarquer que notre méthode fonctionne correctement car peut importe notre position de caméra de départ la trajectoire des points est corrigé pour compenser le mouvement. Dans nos exemples nous avons pris un **nu** de 0.01 ce qui implique les mouvements en tourbillon présent autour de la position souhaitée.

Nous pouvons de plus observer que dû à notre ajout d'erreur cumulé la loi de commande utilisant la matrice d'interaction évaluée à la position courante ne peux plus fournir une trajectoire en ligne droite sauf si nous prenons un paramètre **nu** très bien adapté à la situation. De plus, les minimas locaux sont toujours présent comme nous le montre le cas de rotation à 180°, ce qui est normale puisque la loi de commande n'a pas changé. Sachant que la caractéristique du mouvement des points en ligne droite est dure à paramétrer et les singularités de cette loi de commande sont toujours présentes, la loi de commande évaluée selon la position désirée semble de meilleurs qualitée dans notre exemple.

CONCLUSION :

Pour conclure ce compte rendu de tp, nous avons pu voir implémenter différentes loi de commande utilisant soit des caractéristiques 2D, soit des caractéristiques 3D. Les loi de commandes n'utilisant que des caractéristiques 2D peuvent être intéressantes à utiliser notamment au non-besoin de connaissance 3D de la scène. Il peut être aussi intéressant d'utiliser une loi de commande utilisant une matrice d'interaction évaluée avec la position courante d'un objet si l'on souhaite obtenir une ligne droite sur notre vue de caméra. Cependant il faut faire attention à ne pas tomber dans des cas de singularité tel que la rotation Z de 180°. La loi de commande utilisant une matrice d'interaction évaluée avec la position désirée est un bon compromis si l'on ne souhaite pas forcément obtenir une trajectoire en ligne droite de l'objet et permet d'éviter les cas de singularité. Nous pouvons de plus ajouter que la poursuite d'un objet mouvant se fait sans trop de soucis rien qu'en utilisant un algorithme simple de compensation tel que implémenté dans notre tp.

Nous avons aussi implémenté une loi de commande utilisant des caractéristiques 3D de la scène permettant d'obtenir une trajectoire de la caméra en ligne droite ce qui peut être intéressant dans certains cas. Enfin, nous pouvons noter qu'il existe d'autre loi de commande que nous n'avons pas vu dans ce tp, notamment des méthodes utilisant des caractéristique 2D et 3D qui en même temps qui permettrait d'obtenir des trajectoires en ligne droite de la caméra et de l'objet ce qui entraînera bien sûr des singularités à éviter.