GLO-4001/GLO-7021 Introduction à la robotique mobile
TP1 Version 1.0
Date de remise : 25 février 2022 à 23h55

En équipe de 1 à 2.

11 février 2022

Attention ! N'oubliez pas d'attacher le code de toutes les questions dans la remise .zip de votre travail. Si les codes sont manquants, nous pourrons retirer jusqu'à 20% de la note.

Pour tous les étudiants, vous devez fournir un rapport en un seul document (format pdf) et les fichiers Python zippé. Pour toutes les questions, n'oubliez pas de mettre le détail des calculs.

Pour les étudiants en GLO-7021, veuillez noter qu'une présentation déficiente dans le rapport (manque de clarté, orthographe et grammaire, police de caractère illisible sur figure matlab, etc) pourra entraîner une pénalité allant jusqu'à 10 % de la note. Le rapport doit aussi obligatoirement être formaté avec LaTeX.

Le travail est noté sur 80 pour les étudiants de GLO-4001, et sur 100 pour les étudiants de GLO-7021.

Nous acceptons les réponses en matlab ou Python.

1 Imagerie stéréo (20 pts)

Voici une paire d'images en stéréo à la Fig 1 pour une scène composée de 4 tours parfaitement verticales. Le baseline b entre les caméra est de 7 cm. La tour verte fait 12 cm de hauteur et est située à une distance en z (selon l'axe optique) de 105 cm. L'axe des x de la caméra pointe vers la droite dans les images. Le point principal est $(0, 0)$.

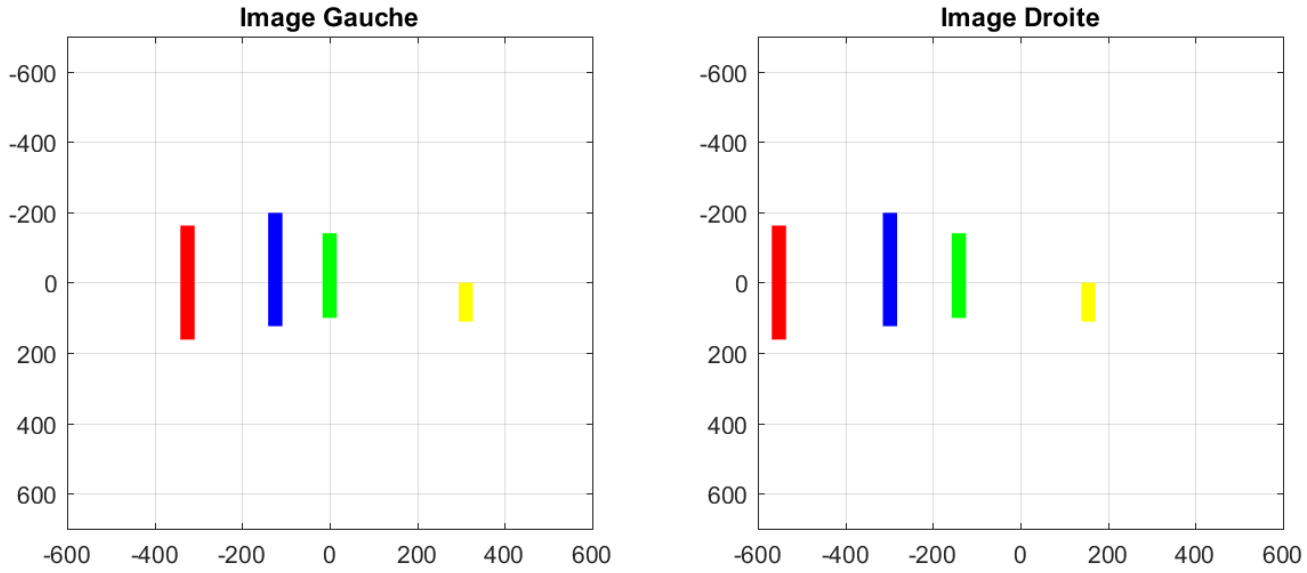

FIGURE 1 – Paire d'images en stéréo. Les coordonnées sont en pixels.

1.1 Distance focale f (5 pts)

Estimez la distance focale f (en pixel) de la caméra, en utilisant la tour verte dans l'image de gauche. Donnez le détail de votre calcul.

1.2 Estimation de la distance A_z de chaque tour (8 pts)

En mesurant la disparité d entre les centres des tours trouvez leur coordonnée A_z respective. Bien indiquer vos mesures et calculs dans votre rapport, et présentez votre réponse finale dans un tableau comme la Table 1.

TABLE 1 – Distance en z des centres des faces des colonnes.

tour	A_z
rouge	
vert	
bleu	
jaune	

1.3 Estimation de la coordonnée A_x de chaque tour (7 pts)

Retrouvez la coordonnées A_x de chacun des objets par rapport à la caméra de gauche. Faites l'hypothèse que l'axe X de la caméra gauche pointe vers la droite, et que le point principal est situé au centre de l'image.

TABLE 2 – Coordonnée en x des centres des faces des colonnes.

tour	A_x
rouge	
vert	
bleu	
jaune	

2 Extracteur de coin FAST (15 pts pour GLO-4001, 18 pts pour GLO-7021)

2.1 Fonction d'extraction des coins FAST (6 pts)

Codez une fonction d'extraction de coins basée sur la méthode FAST, pour des cercles tels que montré à la Figure 2. Notez que vous cherchez un coin ayant 12 pixels continus sur l'arc qui satisfassent le critère. La fonction a le prototype suivant :

```
def detection_coin_FAST(image, centre, seuil)
    return is_fast_corner, intensite_coin
```

où les arguments en entrée sont :

- `image` une image (en noir et blanc) ;
- `centre` un vecteur donnant la coordonnée x-y pour laquelle tester la présence d'un coin dans l'`image` ;
- `seuil` le seuil t, tel que spécifié dans les équations de l'acétate 149 de `03-VisionII.pdf` ;

et les sorties sont :

- `is_fast_corner` indique la présence (1) ou l'absence (0) d'un coin ;
- `intensite_coin` donne l'intensité du coin représenté par la somme V, telle que spécifiée à l'acétate 150 ;

		16	1	2		
	15				3	
14						4
13			C			5
12						6
	11				7	
		10	9	8		

FIGURE 2 – Position des pixels à tester pour le détecteur de coin FAST. Les numéros sont à titre indicatif.

Bug subtil en matlab ! Si vous travaillez avec des UINT8 (ce que fait matlab naturellement pour des images), les valeurs négatives ne seront pas admises ! Il vous faudra donc stocker les intensités lumineuses avec des entiers SIGNÉS.

2.2 Test de votre fonction `detection_coin_FAST` sur une image réelle (9 pts)

Vous allez maintenant tester votre détecteur de coin une image réelle, à partir d'un jeu de données qui a été capturé sur l'île de Devon dans l'arctique canadien par un robot mobile de l'équipe du Prof. Barfoot (U. Toronto). Cette île est reconnue mondialement pour sa similarité avec le sol martien, et sert donc souvent de lieu d'essai pour la robotique interplanétaire. L'image en question est `bw-rectified-left-022148small.png`. Parcourez toute cette image (sauf la bordure à l'intérieur de 8 pixels [1]) et trouvez tous les coins, avec la valeur de `Seuil` de 10 (voir [2]). Pour chaque coin trouvé,

1. Car il ne sera pas possible d'extraire les features BRIEF de la question suivante pour les bordures.
2. Les intensités dans l'image sont entre 0 et 255, car codé en entier non-signé de 8 bits (`uint8`).

marquez sa position à l'aide d'un petit cercle rouge. Attention ! Certaines fonctions d'affichages des images intervertissent les axes x-y. Répondez aux questions suivantes :

— Combien de coins trouvez-vous dans cette image ?
— Quel pourcentage des pixels sont donc considérés comme des coins ?
— Pourquoi cette scène génère-t-elle ce nombre de coins ?
— Quelles régions de l'images contiennent plus de coins ?
— Moins de coins ?

Incluez aussi dans votre rapport l'image avec les coins trouvés.

2.3 Analyse des résultats (GLO-7021 seulement) (3 pts)

Afin de limiter le nombre de coins à traiter, il est possible d'utiliser un heuristique simple, qui permet de ne conserver que les coins les plus forts, basé sur la valeur `intensite_coin`. Cependant, pour utiliser un tel heuristique, il est important de comprendre la distribution de ces valeurs. Dans un histogramme, montrez la distribution des intensités, pour les coins trouvés. Rapportez cette distribution dans votre rapport, et commentez-là.

3 Descripteur BRIEF (19 pts pour GLO-4001, 26 pts pour GLO-7021)

Le descripteur binaire BRIEF est de plus en plus utilisé pour décrire des points de repères naturels dans des problèmes de localisation par caméra. Cette popularité grandissante est en partie dûe à sa facilité de codage, sa robustesse et sa rapidité de calcul. Dans cette question, vous allez explorer l'utilisation de ces descripteurs BRIEF, extraits autour de coins FAST. Pour vous aider, référez-vous au besoin à l'article original du BRIEF : `http://www2.ift.ulaval.ca/~pgiguere/cours/IntroRobotique/notes/calonder_eccv10.pdf` .

3.1 Fonction calculant un descripteur BRIEF (3 pts)

En matlab ou python, écrivez une fonction `ExtractBRIEF(ImagePatch,BriefDescriptorConfig)` qui accepte une patch d'image noir et blanc de SxS pixels avec S=15, ainsi qu'une structure de données `BriefDescriptorConfig`. Cette fonction retourne le descripteur utilisant `BriefDescriptorConfig`, dans un format de votre choix. Cette fonction ne devrait être que quelques lignes de code. Plus d'information sur `BriefDescriptorConfig` est disponible à la question suivante.

3.2 Pipeline d'extraction de features sur une paire d'images réelles (9 pts pour GLO-4001, 11 pts pour GLO-7021)

L'extraction de features visuels est en général une opération coûteuse du point de vue calcul. Ainsi, il est préférable de ne se concentrer que sur des points intéressants dans les images, les *keypoints*. À la question 2, vous avez justement codé une fonction permettant l'extraction de keypoints. Un pipeline d'extraction ressemble typiquement au diagramme de la Figure 3.

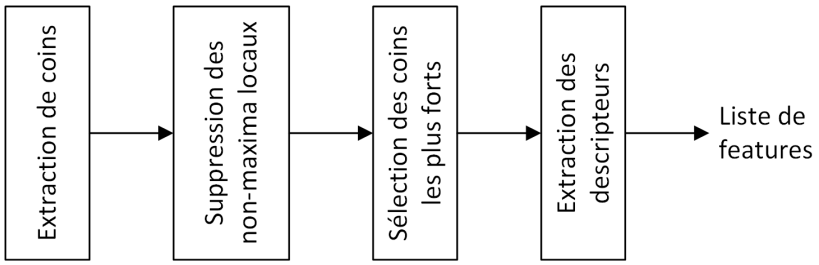

FIGURE 3 – Pipeline d'extraction des features, utilisé lors de la recherche de points de repères naturels dans des images.

Implémentez ce pipeline, en utilisant vos deux fonctions `detection_coin_FAST` et `ExtractBRIEF` pour la première et quatrième étape du pipeline. Pour les étudiants en GLO-4001, la suppression des non-maxima locaux est optionnelle. Pour les GLO-7021, implémenter une version approximative de cette suppression, de votre choix. La sélection des coins sera basée sur la valeur de leur intensité `intensite_coin` : vous ne conserverez que ceux dans le 90ème percentile (autrement dit, les 10 % les plus forts) [3] . Pour les descripteurs, utilisez `numberOfBits`=200 bits. Dans ce programme, la structure de données `BriefDescriptorConfig` qui décrit les `numberOfBits` paires de pixels testées n'est

3. Une manière facile pour identifier ces coins est de trier la liste des coins selon l'intensité, et de ne conserver que un dixième de cette liste.

initialisée une seule fois. Pour générer ces paires de pixels (obligatoirement de manière aléatoire), utilisez une distribution uniforme (`rand()`) sur SxS, avec S=15. Ne vous préoccupez pas des doublons possibles sur ces paires de pixels, pour vous sauver du temps de codage. Assurez-vous que ces paires sont des entiers, via la fonction `ceil()`. Le descripteur doit être extrait dans une fenêtre centrée sur le coin FAST.

Pour les étudiants en GLO-7021, commentez sur :

— votre stratégie pour la suppression des non-maxima locaux ;

— le pourcentage des coins retirés

3.3 Appariement features image gauche-droite (7 pts)

Appliquez votre pipeline sur les images suivantes :

— `bw-rectified-left-022148small.png` et

— `bw-rectified-right-022148small.png`,

disponibles dans le répertoire inclus avec ce TP. Ces images sont issues d'une caméra stéréo. Pour chaque descripteur de l'image de gauche, trouvez le feature dans l'image de droite qui a la plus petite distance de Hamming, donc le plus semblable. Ceci constituera l'appariement. Dans dans votre rapport, mettez une image représentant ces matches. Le fond sera l'image de gauche, et chaque ligne verte reliera le feature de gauche au feature de droite associé, comme dans l'image 4. Si vous avez trop de lignes, choisissez-en un nombre raisonnable de manière aléatoire (une centaine) pour affichage.

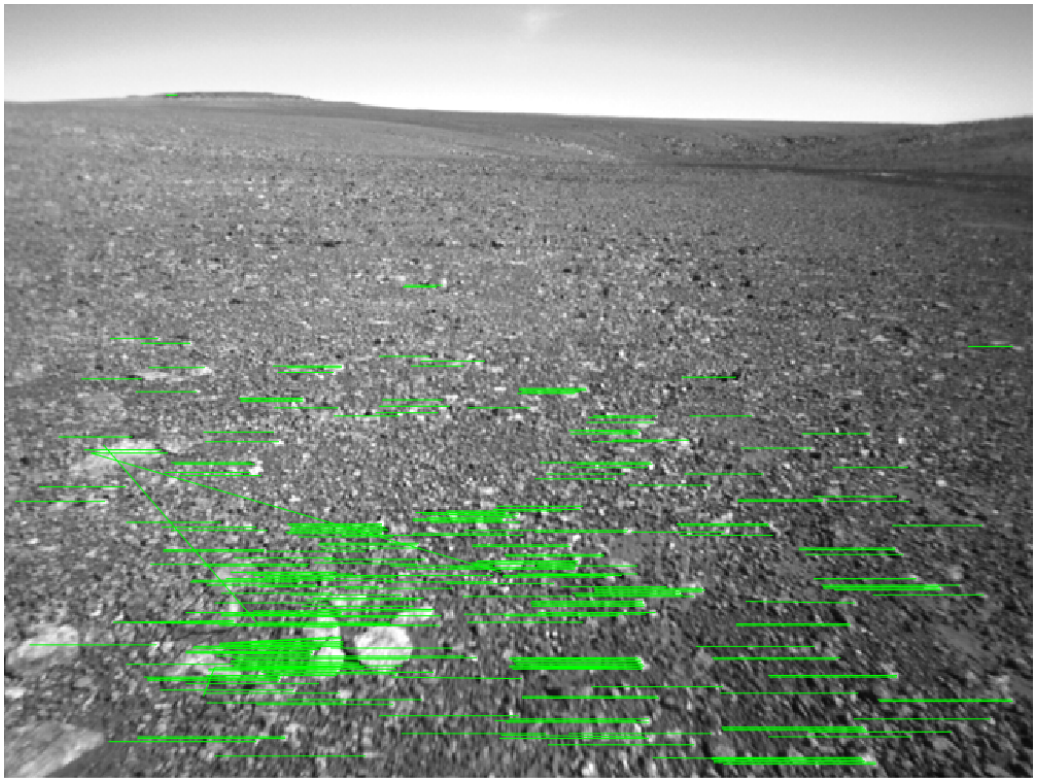

FIGURE 4 – Image similaire à celle que vous devez utiliser, et montrant quelques appariements entre les features de l'image gauche (en fond) vers les features de l'image droite. Notez que j'ai épuré ces matches, afin de ne conserver que de très bons. Vous devriez avoir plus de lignes diagonales dans votre résultat.

3.4 Amélioration de la qualité des matchs (GLO-7021 Seulement) (7 pts)

Vous devriez constater que vous avez beaucoup de faux matches dans l'image. Tentez d'améliorer la qualité de ces matches, en utilisant deux stratégies, soit celles vues en classe, soit de votre crue. Commentez sur l'amélioration apportée par chacune de vos stratégies.

4 Modèle de caméra et positionnement par caméra (26 pts pour GLO-4001, 36 pts pour GLO-7021)

Ces exercices serviront à vous familiariser à la fois au processus de génération d'images et de localisation par caméra, ainsi qu'aux concepts de matrices intrinsèques/extrinsèques de caméra. Dans un premier temps, vous aller devoir faire le code permettant la génération d'images pour un monde simplifié. Dans un deuxième temps, vous allez utiliser du code pour vous localiser par minimisation d'erreur de reprojection. Finalement, à l'aide de nombreuses simulations, vous allez être à même de voir l'impact de plusieurs paramètres du problème (position de la caméra, position des repères, nombre de repères) sur l'incertitude de la relocalisation.

4.1 Génération d'une image (6 pts)

Vous avez trois points de repère, situé au coordonnées *globales* suivantes (en m) :
— L_1 : [-0.2, 0 , 1.2]
— L_2 : [0, 0 , 1]
— L_3 : [0.2, 0, 1.2]
Pour le repère global, l'axe des y pointe en s'éloignant de vous. L'axe des x pointe vers la droite et l'axe des z vers le haut. Le tout est illustré à la Figure 5. Vous allez avoir une caméra dont l'axe optique est parallèle au plancher, et l'axe des y pointe vers le bas (comme pour le repère global). La distance focale de la caméra est de $f = 1000$ pixels.

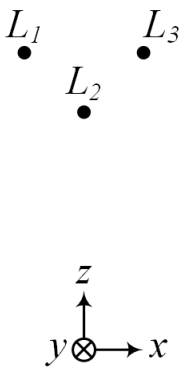

FIGURE 5 – Diagramme illustrant la position des points de repères par rapport au repère global.

En Python, faites une fonction `reprojection()` qui prend la photo :

```
def reprojection(H, focale, L):
```

c'est-à-dire qui calcule la positions des trois points de repère L_i dans les coordonnées u et v du plan image de la caméra. Cette fonction accepte aussi la focale de la caméra et la pose `H` de la caméra dans le repère globale, exprimée par une matrice de transformation homogène 4x4. Notez que selon la convention choisie, l'axe v du plan image pointe vers le bas et l'axe u vers la droite. Incluez dans votre rapport les coordonnées images obtenues.

Pour faciliter la lecture de votre code (et pour vous-même), utilisez des Numpy arrays et l'opérateur de multiplication de matrice `@` dans cette fonction. **Attention !** La matrice `H` est la pose de la caméra, exprimée dans le repère global. Ce n'est donc pas la matrice extrinsèque. Cette dernière devra donc être calculée par vous.

En plaçant la caméra à l'origine (donc, l'axe optique de la caméra pointe dans la même direction que l'axe des z), prenez une photo. Rapportez les coordonnées dans l'images des trois points de repères L_i dans votre rapport, et sauvegarder ces valeurs dans `C`, qui deviendra votre image, dite *cible*, pour la question suivante.

4.2 Localisation par minimisation de l'erreur de reprojection (9 pts)

Une manière de pouvoir vous localiser à l'aide d'une caméra consiste à minimiser l'erreur de reprojection, qui se basera sans grande surprise sur la fonction `reprojection()`. La pose de la caméra minimisant l'écart entre les points de repères dans l'image cible `C` et ceux calculés par la fonction de reprojection sera considérée comme la bonne. Pour trouver cette pose, vous allez devoir utiliser la fonction d'optimisation `fmin`[4] :

```
from scipy.optimize import fmin
```

Pour simplifier le tout, nous allons aussi faires des hypothèses simplificatrices. Ainsi, la caméra ne pourra se déplacer qu'en trois degrés de libertés seulement. Conséquemment, les seuls paramètres de la pose de la caméra qui seront optimisés par `fmin` lors de la recherche de la pose seront les suivants :

- la position en x ;
- la position en z ; et
- l'angle θ de rotation (autour de l'axe y). Un angle $\theta = 0$ signifiera que l'axe optique est parallèle à l'axe des z du repère global.

La localisation consistera donc à résoudre la minimisation de l'erreur de reprojection au carré pour tous les points de repères L_i. L'erreur de reprojection (dit *résiduel*) r_i, pour un point de repère L_i imagé est la distance euclidienne suivante :

$$r_i = ||c_i - f_{reprojection}(L_i)||, \tag{1}$$

où c_i représente la position (en pixel) du repère i dans l'image cible. Notez que la fonction $f_{reprojection}$ dépendra des paramètres intrinsèques et extrinsèques de la caméra, que j'ai omis ci-haut pour alléger la notation. La recherche de la pose de la caméra sera donc la minimisation suivante :

$$\underset{x,y,\theta}{\operatorname{argmin}} \sum_{i \in Repere} r_i^2. \tag{2}$$

La fonction à minimiser (à implémenter) sera :

```
def somme_des_residuels_au_carre(pose_camera,focal,L,C):
```

où `pose_camera` sera un vecteur contenant les valeurs $[x, z, \theta]$ de la pose de la caméra, exprimé dans le repère monde, `C` contient les coordonnées de tous les repères $\mathbf{L}$ dans l'image cible, `L` contiendra les coordonnées 3D en homogènes des points de repères dans le repère global, et `focale` sera la focale de la caméra.

L'optimisation se fera avec la commande suivante :

```
pose_solution = fmin(somme_des_residuels_au_carre, pose_initiale_camera,
                     args=(focal, L, C), maxiter=1000)
```

4. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin.html

La `pose_initiale_camera` sera un vecteur contenant les valeurs estimées $[x, z, \theta]$ de la pose de la caméra. Notez ici qu'ils correspondent aussi aux variables libres dans l'optimisation. Dans votre cas, initialisez ce vecteur à un endroit légèrement décalé de la position réelle, soit d'environ 0.2 unités pour x et z et d'environ 0.2 rad pour θ. Dans votre rapport, indiquez :

— le nombre d'itérations effectués par `fmin` ;

— la solution trouvée et son écart par rapport à la vraie pose $[x, z, \theta] = [0, 0, 0]$.

4.3 Impact du bruit sur l'estimation des repères via une approche de type Monte Carlo (10 pts pour GLO-4001, 7 pts pour GLO-7021)

En réalité, la détection de point de repères dans une image ne se fait pas parfaitement. Ainsi, il y aura une petite erreur entre la position estimée et la position réelle, en pixel, de ces repères. Vous allez simuler cela en ajoutant du bruit sur la coordonnée en u de ces repères. Nous n'ajoutons pas de bruit en v car nous ne sommes intéressés que par la position horizontale de ces repères dans l'image. Ce bruit suivra une distribution gaussienne, avec écart-type $\sigma_p = 2$ pixel.

Montrez la distribution des estimés de pose de la caméra sur une carte 2D (x, z), en traçant un point par estimé trouvé par simulation. Chaque simulation consiste simplement à piger les bruits aléatoirement, de les additionner au valeurs véritables des u, puis de retrouver la pose avec la méthode de la section précédente. Répétez cette routine mais pour des nouvelles poses de caméras en la reculant de 1 à 7 m, par incrément de 1 m.

Pour chaque pose, faites 1000 simulations bruitées et marquez d'un point sur un graphique la pose trouvée, afin de visualiser approximativement cette distribution. Par exemple, la Figure 6 montre un ensemble hypothétique de distributions illustrées de la manière demandée, pour des poses (entre 0 et 7 m) comparables à l'exercice demandé.

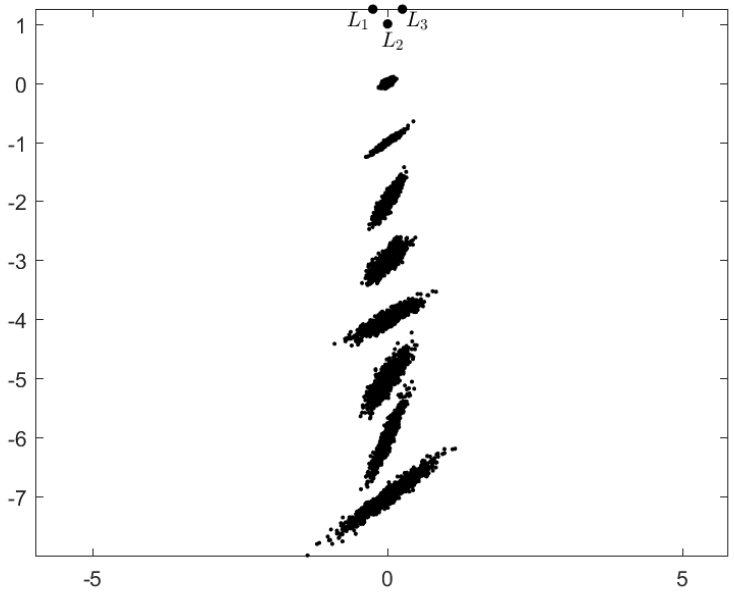

FIGURE 6 – Exemple de distributions quelconques (non, ce n'est pas la solution !) pour différentes poses de la caméra. Notez la position fixe des repères L_i, tel que demandé dans l'exercice, et le centre des distributions qui reculent, selon z_{carte}.

4.4 Discussion sur la forme de la distribution (3 pts) GLO-7021 seulement

Calculez la covariance empirique de cette distribution pour chacune des poses. Que remarquez-vous sur la forme de ces distributions, en particulier les termes en dehors de la diagonale de cette matrice de covariance ? À quoi attribuez-vous cela ?

Tracez une figure montrant les écart-types σ_{xcarte} et σ_{ycarte}, en fonction des huit poses réelles (de 0 à 7 m). Discutez.

4.5 Influence de la position des points de repère sur la qualité de la localisation (3 pts) GLO-7021 seulement

Doublez l'espacement des points de repère, tout en conservant la même position pour L_2 et la même configuration (en triangle) des trois points de repères. Refaites les expériences des sections 4.3 et 4.4. Commentez sur l'impact de l'incertitude de localisation, en fonction de cette augmentation de la distance entre les repères.

4.6 Influence du nombre de points de repère sur la qualité de la localisation (7 pts) GLO-7021 seulement

Une règle de base que l'on a vu en classe est que l'incertitude d'un estimé diminue généralement en $1/\sqrt{n}$, où n est le nombre de mesures utilisées. En dispersant aléatoirement n points de repères devant la caméra en suivant une distribution normale entre -0.5 et 1.5 m en z et -0.5 et 0.5 m en x, estimez la précision de la localisation. Réutilisez votre code de la question 4.3.

Commentez en un paragraphe l'impact du nombre de points de repères n sur l'incertitude de localisation. Justifiez le choix des n dans votre expérimentation. Ajoutez des graphiques montrant l'écart type en x et en z de la localisation, en fonction de n.