Introduction à la robotique mobile GLO-4001/GLO-7021
Philippe Giguère A14

Laboratoire N^{o} 4

Caméra

Travail en équipe de 2

Matériel fourni : règle alu 1m, règle plastique 30 cm, rapporteur d'angle, marqueurs fiduciaires, repère vertical (Lego).

Partie 1 – Calibration de la longueur focale

Installez la caméra sur le robot. Fixez la longue règle métallique sur la table à l'aide de ruban adhésif, dans le sens de l'axe optique de la caméra. La caméra doit être centrée vis-à-vis la marque de 0 cm de la règle. Fixez la règle de 30 cm perpendiculairement à celle déjà en place, à une distance d'environ A_Z=60 cm, et idéalement à la même hauteur que la caméra sur le robot. Vous pouvez utiliser une surface comme une feuille de papier derrière la règle afin de mieux discerner les divisions de mesure. Sur l'image ici-bas, la caméra est montée sur un support en plastique noir, mais cela vous donne une idée de l'assemblage désiré.

Vous allez utiliser ros4mat pour prendre une photo. Le script **`SetupCamera.m`** va configurer la caméra pour des images 640x480, avec 2 images par secondes. Assurez-vous de bien configurer l'adresse IP dans ce script **`SetupCamera.m`** .

Pour prendre une image, ne conserver que la dernière, et l'afficher à l'écran, faites les commandes suivantes :

```
image = ros4mat('camera');
LastImage = image(:,:,:,end); % Car c'est une matrice d'image que ros4mat retourne
imagesc(LastImage);
```

Si l'image qui apparait provient de la caméra de votre laptop, il vous faut ajuster le paramètre **`CameraID`** à 1 dans **`CameraSetup.m`**. Si les images apparaissent trop sombres lors de la capture, ajustez le paramètre `Exposure` du même fichier.

Si vous avez le message d'erreur suivant à l'affichage de l'image :

```
Index exceeds matrix dimensions.
Error in CaptureImage (line 9)
    imagesc(image(:,:,:,1));
```

C'est que l'image retournée était vide. Faites alors simplement une deuxième capture d'image.

Avec la commande suivante, vous pouvez cliquer à l'écran pour récupérer les coordonnées pixels de des deux divisions séparées de 30 cm sur la règle de plastique :

```
[Lx Ly] = ginput(2);
```

Assurez-vous que la règle apparaît au centre de l'image (horizontalement et verticalement), sans quoi le résultat obtenu sera moins bon. Calculez, en utilisant le théorème de Thalès, la valeur de la focale *f* de la caméra, en pixel :

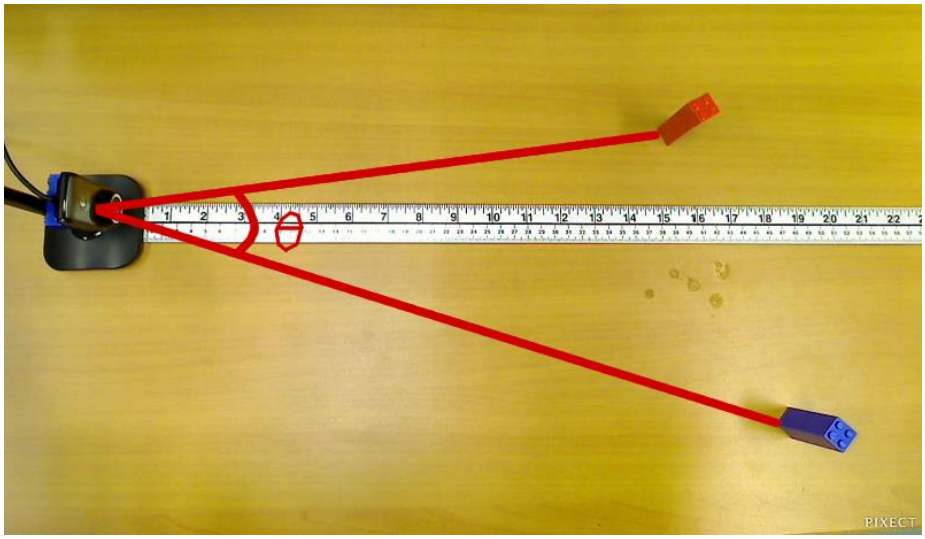
$$f = \Delta L_x \frac{A_z}{30}$$

où $\Delta L_x = L_x(1) - L_x(2)$, avec $L_x(2) > L_x(1)$ (donc cliquez de gauche à droite dans l'image), et le 30 correspond à la distance entre les marqueurs 0-30 cm sur la règle.

Vérifiez que votre calibration est adéquate, en déplacez la règle de plastique à une distance différente de la caméra et prenez une nouvelle photo. Toujours en utilisant le théorème de Thalès, calculez cette fois-ci la distance A_Z de la règle en utilisant la valeur de la focale *f* que vous avez déterminée (simple règle-de-trois avec l'équation ci-dessus). Comparez ensuite la distance A_Z trouvée avec la distance réelle.

Partie 2 –Mesure d'angles

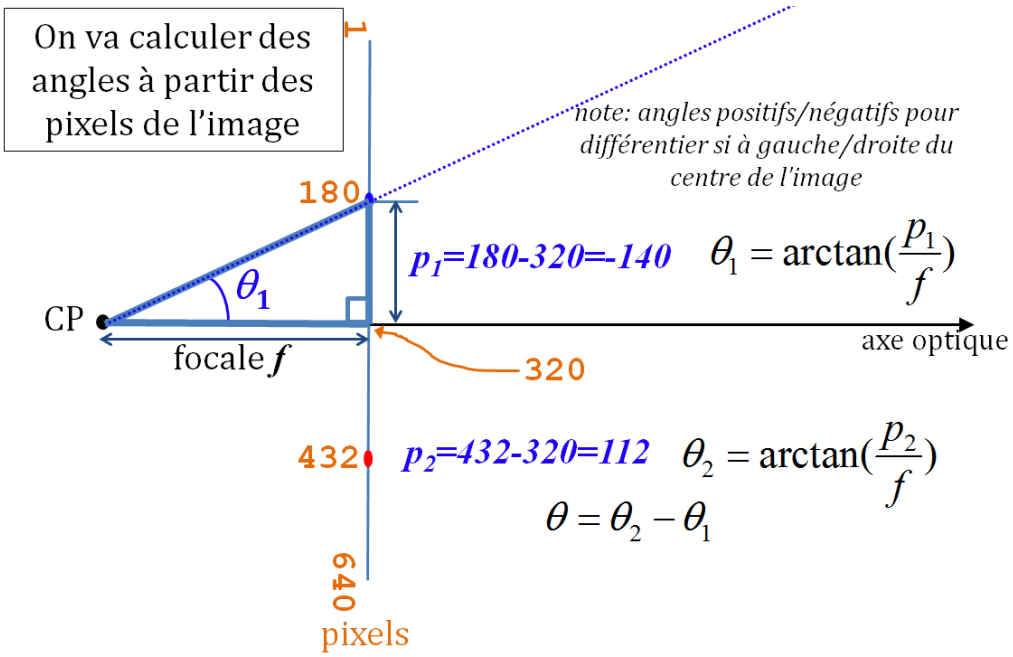
Comme mentionné en classe, une caméra est un rapporteur d'angle, en quelque sorte. Placez deux objets dans le champ de vision de la caméra (comme les blocs rouges et bleus). En vous servant des règles et d'un rapporteur d'angle, mesurez l'angle entre ces deux objets, du point de vue de la caméra.



Capturez une image avec la caméra et déterminez les coordonnées pixels (x seulement) du centre des objets (horizontalement), avec la commande

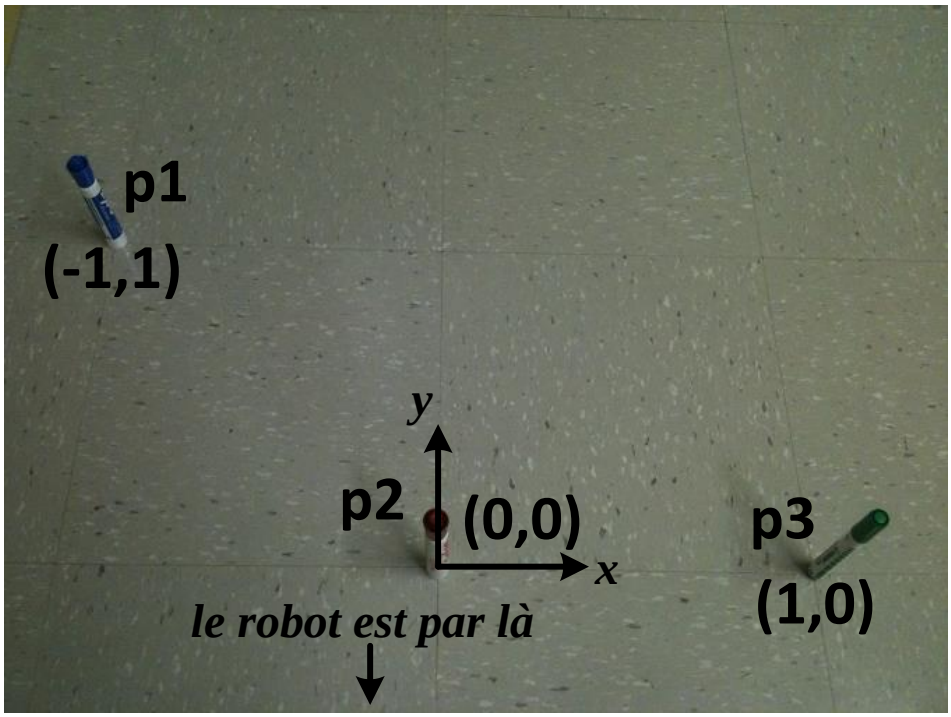
```
[Lx Ly] = ginput(2);
```

Calculez l'angle θ entre les objets à partir des coordonnées images L_x trouvées. Pour ce faire, vous devez considérer que le centre optique de la caméra passe par le centre de l'image (ici, 320/2 = 160). Comparez la valeur obtenue avec la valeur réelle. Note : utilisez la fonction **`atan2(y,x)`** dans Matlab pour calculer l'arc tangente, nécessaire au calcul d'angle tel que montré ici-bas.

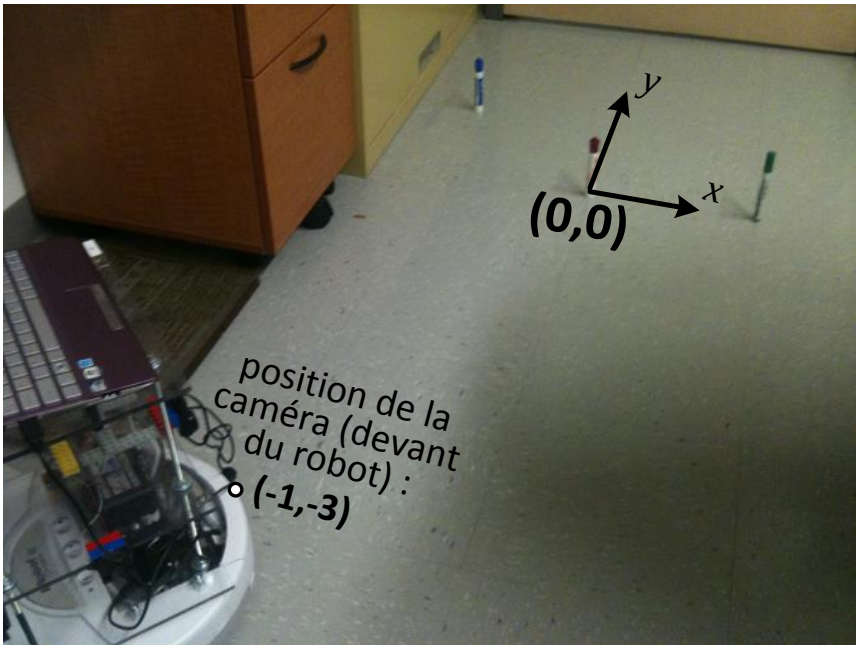


Partie 3 – Localisation en deux dimensions

Placez cette fois trois objets sur le plancher, en vous servant des tuiles de plancher comme système cartésien. Le point de repère p2 sera la position (0,0) de votre référentiel monde. Ce référentiel aura comme unité de longueur un *tuilex*, unité imaginaire correspondant à un côté d'une tuile (environ 30 cm). Placez les deux autres objets sur des intersections de coins de tuiles, comme illustré ici-bas, avec les coordonnées dans les unités *tuilex*.



Placez la caméra du robot de sorte qu'elle soit verticalement au-dessus d'un coin de tuile, pour que vous puissiez facilement connaitre sa vraie coordonnée et la comparer avec l'algorithme. Aussi, il faut que le robot soit situé dans les *y* négatifs (question de codage, pas une limitation de la technique).



Modifiez le script **`PositionCamera.m`** pour inclure votre focale *f*. Si vous choisissez des endroits différents pour p1, p2 et p3, vous devez modifier ces valeurs aussi.

Exécutez maintenant ce script **`PositionCamera.m`**. Attention! Il est important que l'ordre des clics de souris sur l'image corresponde aux repères p1-p3. Ainsi, le premier clic se fera sur p1, le deuxième sur p2, et le troisième sur p3. Si vous ne respectez pas cela, le code vous retournera n'importe quoi comme position.

Ce script effectue essentiellement ces calculs :

- coordonnées du centre d'un cercle :

```
c = [0 -1; 1 0]*(p1-p2)./(2*tan(alpha))+ 0.5*(p2+p1);
où p1=[x1  y1]' et p2=[x2  y2]' (vecteurs colonne)
```

- rayon d'un cercle :

```
r2 = d2 / (2 * sin(alpha));
```

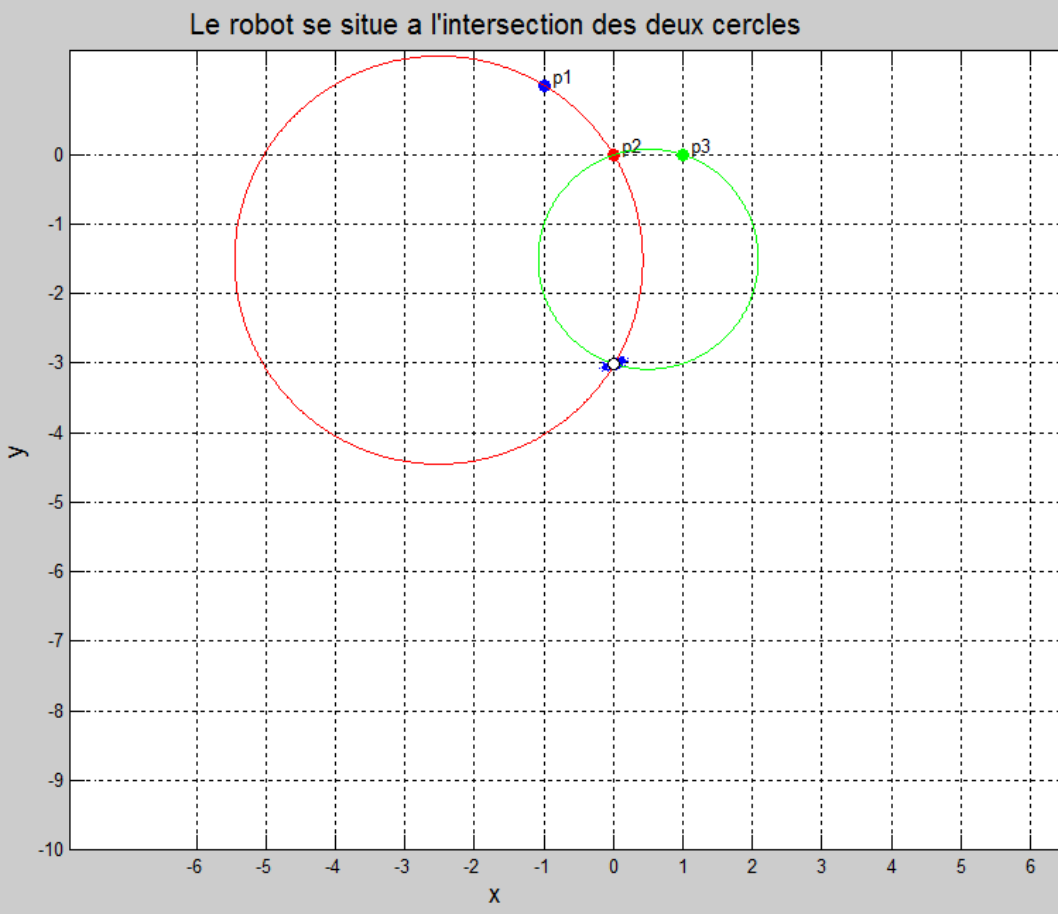
Partie 4 – Impact du bruit sur la localisation

Le script **`PositionCameraAvecBruit.m`** sert à vous monter l'effet du bruit de mesures sur la localisation du robot. Vos mesures ici sont les positions en x des repères dans l'image, comme dans la partie précédente. Le code va ajouter du bruit sur ces mesures en pixel, avec une distribution normale d'écart-type **`spixel`**. En simulant un millier de mesures bruitées, ce code sera en mesure de voir la distribution des poses possibles du robot, pour les mesures que vous avez effectuées. Les mesures bruitées sont simulées comme suit dans le code :

```
NoisyLx = Lx + spixel*randn(3,1);
```

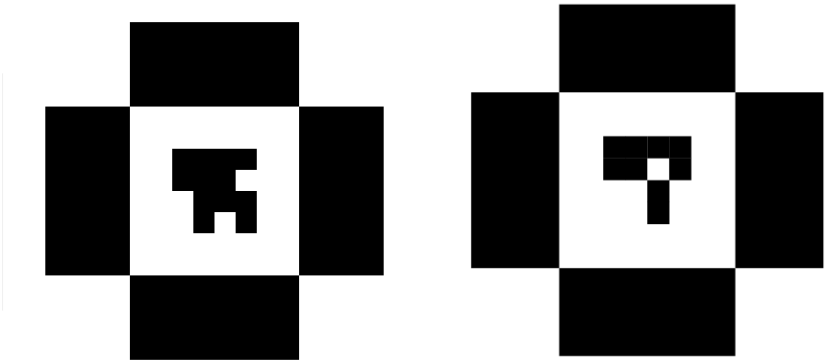
La position estimée (avec le bruit) pour chaque simulation est retrouvée grâce à la commande `circleintersect`. Pour mieux comprendre, rien de mieux que de regardez le code.

Modifiez le script **`PositionCameraAvecBruit.m`** pour inclure votre focale *f*, et ajuster si vous le désirer la valeur d'écart-type **`spixel`**. Exécutez le code (qui fonctionne essentiellement comme à la section précédente) qui vous demandera de cliquer sur les repères dans l'image. Vous devriez voir la figure ici-bas, après une pause de quelques secondes. Les petits points en bleus sont le résultat de la localisation simulée bruitée. Notez comment cette distribution change de forme, en fonction de la position du robot. En effet, vous devriez voir sa taille s'accroître avec la distance entre le robot et les repères, indiquant ainsi que la précision du système décroit avec la distance.



Partie 5 – Détection automatique des repères (facultatif)

Comme vous avez pu le constater, les scripts précédents exigent une intervention humaine pour la localisation des marqueurs visuels dans l'image. Bien évidemment pour un vrai robot, cette solution serait inacceptable. Une façon de contourner ce problème est de placer dans l'environnement des marqueurs appelés fiduciaires, qui sont facilement détectables et identifiables par des programmes de traitement d'image. Chaque marqueur contient un code binaire l'identifiant. Ces marqueurs sont déjà imprimés pour vous.



Le code `Caltag` et `CaltagHack_4001_7021.zip` sur le site web de cours vous permettent d'extraire la position et le numéro de chaque marqueur automatiquement dans une image. Modifiez le script `PositionCamera.m` (éliminer l'étape `ginput`, essentiellement) pour faire la localisation de façon entièrement automatique.