**Formation IBEX**

TP n°3 - **Programmation par contracteurs**

---

Le but de cet exercice est d'effectuer un programme de localisation robuste (dans un contexte de mesures aberrantes), via la programmation par contracteurs.

**Modèle.** Un robot se déplace en deux dimensions sur une durée de $T$ pas de temps (voir figure). Les temps sont notés $t_0, \ldots, t_{T-1}$ et on note $x_t$ la position du robot à l'instant $t$, qui est donc un vecteur à 2 composantes.

Le robot est équippé d'un capteur lui permettant, à chaque instant $t$, de mesurer la distance qui le sépare de toutes les balises présentes dans l'environnement. On note $N$ le nombre de balises et $b_i$ la position de la $i^{eme}$ balise, $0 \leq i \leq N-1$.

Ainsi, à chaque pas de temps, le robot dispose de $N$ mesures de distance notées $d_{ti}$ avec

$$d_{ti} = distance(x_t, b_i), \quad 0 \leq t \leq T-1,\ 0 \leq i \leq N-1.$$

On note enfin $v_t$ le vecteur représentant le déplacement effectué par le robot entre les instants $t$ et $t+1$, c'est à dire,

$$x_{t+1} = x_t + v_t, \quad 0 \leq t \leq T-2.$$

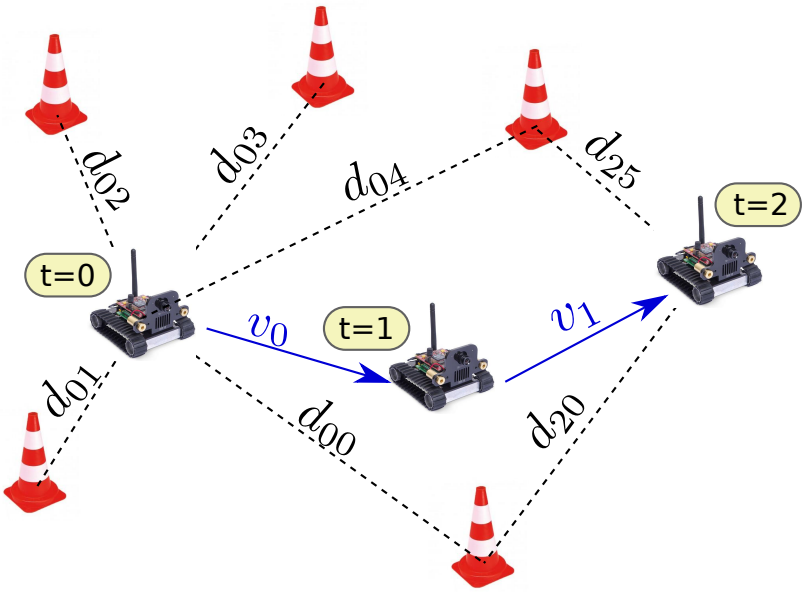


FIGURE 1 – Exemple du problème de localisation robuste avec $T = 3$ pas de temps et $N = 5$ balises (toutes les distances mesurées ne sont pas représentées).

**Incertitudes.** Nous considérons d'une part

1. une absence de toute information a priori sur les positions $x_t$
2. une incertitude sur la position des balises, bornée par une valeur nomée `BEACON_ERROR` dans le programme
3. une incertitude sur les mesures de distances, bornée par la valeur `DIST_ERROR`
4. le fait qu'à chaque pas de temps, au plus `NB_OUTLIERS` mesures peuvent être incorrectes.

En revanche, pour simplifier l'exemple, nous supposons que nous connaissons exactement les vecteurs $v_t$.

**But.** Le but est d'estimer de façon garantie la trajectoire du robot en donnant, pour chaque $t$, une boîte dans laquelle on garantit que la position $x_t$ du robot est incluse.

▶ Téléchargez le fichier `robust_loc.cpp`.

Le fichier ne contient pour le moment que la déclaration des variables et constantes du programme ainsi qu'une fonction `init_data` qui se charge de générer (simuler) toutes les mesures. Vous pouvez **ignorer les détails** de cette fonction.

## 1- Contracteur sur la distance

La classe `Distance` va être le contracteur de base, celui permettant de contracter la position $x_t$ d'un robot à l'instant $t$ par rapport à une balise donnée. Notre souhait est que ce contracteur prenne en entrée une boîte représentant uniquement $x_t$, c'est à dire une boîte de dimension 2. Cela sera justifié par la suite.

Le constructeur de cette classe vous est fourni, détaillons-le. Tout d'abord, notre contracteur prend bien deux arguments : les indices $t$ (temps) et $i$ (numéro de balise). Le fait d'écrire ensuite `Ctc(2)` dans la partie "initialisation" permet précisément d'indiquer que ce contracteur travaillera sur des boîtes de dimension 2. L'initialisation se termine par la mémorisation des valeurs de $t$ et $i$. Passons au code à proprement parler du constructeur. Il est clair tout d'abord que pour contracter la boîte $[x_t]$ suivant la contrainte

$$d_{ti} = distance(x_t, b_i),$$

il nous faut définir la fonction distance. Cette fonction admet 2 vecteurs de dimension 2 (soit 4 variables en tout) et s'écrit ainsi :

$$\texttt{distf} : (p, q) \mapsto \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}.$$

La fonction `distf` permet donc de contracter une boîte représentant 2 vecteurs. Notons cette boîte $[x_t b_i]$. Lorsqu'une boîte $[x_t]$ nous est fournie, il nous faut dans un premier temps construire cette boîte en y adjoignant la boîte $[b_i]$. Pour "écrire" $[x_t]$ dans $[x_t b_i]$ on utilisera le *backward* de la 1ère fonction de projection :

$$(x_t, b_i) \mapsto x_t.$$

et, de même, pour écrire $[b_i]$ dans $[x_t b_i]$ on utilisera le *backward* de la 2nde fonction de projection :

$$(x_t, b_i) \mapsto b_i.$$

▶ Implémentez la fonction `contract` de la classe `Distance`. Cette fonction doit :
- créer une boîte $[x_t b_i]$ dont la taille est 4 (le nombre de variables de la fonction `distf`, qu'on peut obtenir aussi en écrivant `distf.nb_var()`) ;
- "écrit" $[x_t]$ et $[b_i]$ dans cette boîte ;
- la contracte en utilisant `distf` ;
- "lit" $[x_t]$ depuis $[x_t b_i]$ (utilisez le *forward* de la 2nd fonction de projection)

▶ Testez-la (décommentez le test de la Question 1 dans `main`). La réponse attendue est donnée en commentaire.

## 2- Contracteur sur le déplacement

Ecrivez le contracteur `OneMove` (classe héritant de `Ctc`). Pour un $t$ donné, cette classe devra contracter suivant la contrainte

$$x_{t+1} = x_t + v_t.$$

Comme à la section précédente, on pourrait exiger que ce contracteur travaille directement sur une sous-boîte, en l'occurrence celle constitué des vecteurs $x_t$ et $x_{t+1}$, mais il ne sera pas nécéssaire cette fois d'imposer cette restriction. On fera donc en sorte que le contracteur prennent directement la boîte globale des positions, de taille $2 \times T$.

Le principe de ce contracteur est le suivant :

1. lecture de la sous-matrice $(x_t, x_{t+1})$ à partir de la boîte globale représentant $(x_0, \ldots, x_{T-1})$ (utilisez une fonction de projection)
2. transformation de cette sous-matrice en boîte (utilisez une fonction identité)
3. contraction de cette boîte suivant la contrainte précédente
4. transformation inverse pour récupérer une matrice
5. écriture de cette matrice dans la boîte globale

▶ Déclarez les champs et faites le constructeur

▶ Implémentez la fonction `contract`.

▶ Testez-la (recommentez le test de la question 1 et décommentez celui de la question 2). La réponse attendue est donnée en commentaire.

## 3- Q-intersection

A un instant $t$ notre but est désormais de contracter la position du robot en considérant $q = N - \texttt{NB\_OUTLIERS}$ mesures correctes, c'est à dire en prenant l'ensemble des contractions possibles résultant d'un choix de $q$ combinaisons des contracteurs de distance parmi $N$. Mathématiquement, en notant $i_1, \ldots, i_q$ un ensemble de $q$ indices tous différents entre 0 et $N-1$ :

$$\texttt{QInter}(t) = \bigcup_{i_1,\ldots,i_q} \bigcap_{1 \leq j \leq q} \texttt{Distance}(t, i_j)$$

Vous n'aurez pas à faire ces calculs d'union et d'intersection, qui sont déjà effectué par un contracteur prédéfini, nommé `CtcQInter`. Le contracteur `CtcQInter` prend en argument un tableau de contracteurs travaillant tous sur des boîtes de même dimension. En l'occurence, cette dimension sera 2 et le tableau contiendra tous les contracteurs de distance à l'instant $t$.

Le contracteur `CtcQInter` travaillera alors lui-même sur des boîtes de dimension 2 et effectuera le calcul correspondant à la formule ci-dessus.

▶ Créez la class `QInter`.

▶ Déclarez les champs et faites le constructeur. Notre souhait est que `QInter` (notre contracteur) travaille sur la boîte globale (de taille $2T$). Vous aurez donc de nouveau besoin, par la suite, d'une fonction de projection pour extraire $x_t$. Initialisez le champ `q` en créant tout d'abord un tableau de contracteurs de type `Array<Ctc>`, en le remplissant via `set_ref`. Enfin, donnez ce tableau au constructeur de `CtcQInter`.

## 4- Point fixe global

Directement dans le `main` :

▶ créez et remplissez un `vector<Ctc*>` en ajoutant pour tout $t$, $0 \leq t \leq T-1$, le contracteur `QInter`(t) et (sauf pour $t = T-1$) `OneMove`(t).

▶ Créez une composition de tous ces contracteurs via `CtcCompo`.

▶ Faites un point-fixe de cette composition via `CtcFixPoint`.

▶ Testez le point fixe global en affichant la boîte initiale et le résultat de la contraction. Modifiez les valeurs des constantes (`N`, `T`, `NB_OUTLIERS`, etc.) et observez les changements.