BE : Système de navigation multi-senseurs

Damien Vivet

2021

Introduction

L'objectif de ce BE est de mettre en application le Filtrage de Kalman dans le contexte de la robotique autonome et plus précisément de la navigation multi-senseurs.

L'application visée consiste en la localisation et la navigation d'un robot mobile simple dans un monde 2D. Ce même type d'algorithme est utilisé pour la navigation autonome de véhicule comme celui de l'équipe Navigation de la figure 1.

FIGURE 1 – Type de robot mobile utilisant des algorithmes de navigation multi-senseurs

La plateforme mobile est équipée de centrales inertielles, de capteurs GNSS, de caméras stéréo et monoculaires, d'un Laser 3D ainsi que d'un système de positionnement de référence. L'ensemble de ces capteurs est utilisé dans des approches de navigation multicapteur.

Dans le cadre de ce BE, le robot simulé est équipé avec des capteurs proprioceptifs dont des capteurs de vitesses linéaires et angulaire. Un laser 2D est également disponible ainsi qu'un capteur GNSS.

Les objectifs sont :
- Comprendre le principe d'un filtrage de Kalman,
- Mettre en place un modèle d'évolution de véhicule,
- Mettre en place différents modèles d'observation,
- Réaliser un premier algorithme de SLAM simplifié.

Ce BE sera réalisé sur MATLAB en utilisant les fichiers de mesure simulés fournis.

1 Prise en main de l'environnement et des données

Le robot est équipé d'un capteur de vitesse fournissant la vitesse linéaire V, la vitesse angulaire w, ainsi que d'un lidar 2D fournissant une liste d'impacts, chaque impact appelé amer contient les données suivantes : $amer = \{\rho, \theta, ID\}$. Ces capteurs sont tous synchronisés et fournissent des données à la fréquence de $F_s = 1$ Hz.

Il est à noter que la simulation a été réalisée avec des capteurs de deux gammes différentes. Chaque enregistrement correspond à un fichier de données différent dont les variances sont fournies dans le tableau suivant :

— *SLAM_ 1.mat* : données acquises avec des capteurs très bon
— *SLAM_ 2.mat* : données acquises avec des capteurs de qualité moyenne

	σ_V	σ_ω	σ_ρ	σ_θ
Data 1	0.01	0.005	0.05	0.005
Data 2	0.1	0.05	2	0.05

Ces données sont fournies dans les variables Pu et Pz sous Matlab.

2 Approche de localisation et cartographie simultanées

L'objectif est de naviguer dans un environnement inconnu et non-instrumenté. Un exemple de trajectoire avec les objets à découvrir dans l'environnement sont représentés sur la figure 2.

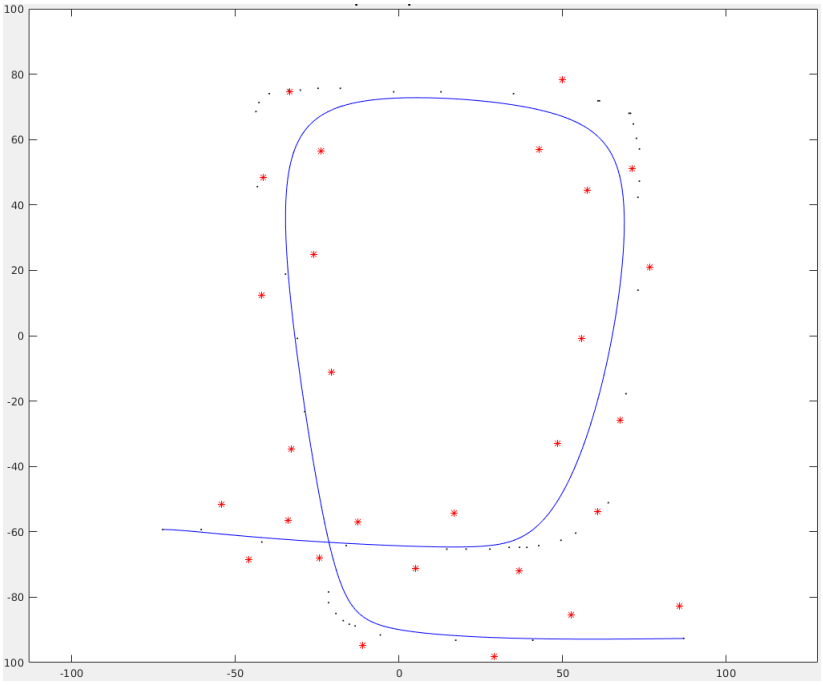


FIGURE 2 – Trajectoire et positions des amers à reconstruire

Chaque point rouge représente un objet de l'environnement qui peut être détecté par notre capteur embarqué. Dans une d'application réelle, ces objets peuvent être sémantiques (panneaux de signalisation, véhicules, poteaux etc) ou géométriques (arrêtes, droites, élément ponctuel, etc).

Modèle d'état du robot

On souhaite à tout instant connaître la position 2D du robot (x, y), ainsi que son orientation θ. De plus on souhaite obtenir la cartographie de l'environnement $\mathbf{m} = \{a_1, a_2, \dots a_k\}$ représentant les différents objets détectés et mémorisés. Estimer les états sous-entend que l'on doit également connaître les covariances de tous ces états. En effet, nous utilisons des données incertaines (erreurs de mesures) et cette incertitude doit être prise en compte pour estimer au mieux l'état du robot.

Ainsi on cherche à chaque point de la trajectoire à estimer :

$$X = \begin{bmatrix} x \\ y \\ \theta \\ \mathbf{m} \end{bmatrix} \text{ et } PX$$

$$\text{avec } \mathbf{m} = \begin{bmatrix} a_{1x} \\ a_{1y} \\ \vdots \\ a_{kx} \\ a_{ky} \end{bmatrix}$$

Capteurs embarqués

Nous utilisons un capteur extéroceptif de type LIDAR 2D. Ce dernier permet d'obtenir des informations sur les éléments présents dans le voisinage direct du robot $z = \{\rho, \theta, ID\}$. $\{\rho, \theta\}$ sont les coordonnées polaire de l'objet dans le repère du LiDAR, ID représente un identifiant unique de l'objet permettant de le différencier des autres. Dans une application réelle, ID doit être créé à partir d'un descripteur suffisament pertinent pour lever toute ambiguité de mise en correspondance.

Travail à réaliser

1. Coder un modèle d'évolution ($X_{t+1} = f(X_t, u_t)$) du véhicule utilisant en entrée les données du capteur de vitesse linéaire V et du capteur de vitesse angulaire w afin de tracer la trajectoire estimée du robot. (Aide en annexe)
2. Réaliser également la prédiction de la covariance de l'état (PX_{t+1} à partir de PX_t).
3. Chaque fois qu'une mesure LIDAR est fournie, les utiliser pour détecter et positionner de nouveaux objets de l'environnement (étape de Mapping). Il faut bien entendu positionner les amers dans le référenciel "monde" et non "robot".

> ***Astuce :*** *où sont stockés ces objets (la carte) ? Les amers étant statistiquement liés à l'état du véhicule, ces amers doivent être ajouté dans le vecteur d'état du robot. On estimera ainsi conjointement la position du robot et la carte de l'environnement : on fait donc du SLAM (localisation et cartographie simultanée).*
>
> $$X = [x, y, \theta, a_{x1}, a_{y1}, \dots, a_{xn}, a_{yn}]^T$$
>
> *Pourquoi garde-t-on les amers dans le vecteur d'état ? En fait, nous devons garder (ou construire) la correlation statistique entre l'amer et la pose du robot de telle façon qu'une nouvelle information sur la pose puisse améliorer l'amer mais aussi qu'une information sur l'amer puisse corriger la pose. Il faut donc construire la matrice de covariance quand on ajoute un amer. Pour cela une solution possible consiste à construire directement la matrice de covariance en utilisant la Jacobienne reliant l'état à l'amer :*
>
> $$a_i = \begin{bmatrix} a_{xi} \\ a_{yi} \end{bmatrix} = g(X_k, z_k)$$

donc

$$PX = \begin{bmatrix} PX & PXJ_{gX}^T \\ J_{gX}PX & C_{a_i} \end{bmatrix}$$

avec C_{a_i} la matrice de covariance de l'amer détecté.

4. Veiller à ce que chaque amer ne soit ajouté qu'une seule fois dans le vecteur d'état. Pour cela il faut conserver la lsite des identifiants des objets déja connus. Pour l'ajout utilisez la méthode décrite ci-dessus.
5. Modifier le programme de façon à ce que chaque fois qu'un amer déjà connu est observé, une mise à jour soit réalisée. Pour cela, il faut premièrement savoir quel objet est vu grace à son identifiant, ensuite, il faut récupérer ses données dans le vecteur d'état. Enfin, on procède à une mise à jour classique du filtre de Kalman. (Aide en Annexe)
6. Comparer les résultats obtenus en fonction de la gamme de capteur utilisée.
7. Commenter et expliquer ce qui est observé.

3 Ajout d'un capteur GNSS

Supposons à présent que le robot soit équipé d'un capteur de positionnement absolu type GNSS. Supposons que le capteur simulé fournisse des mesures à une fréquence bien moins élevée (0.02 Hz) que les autres capteurs et que ses observations soient $x(_{gnss}, y_{gnss})$ associé à une covariance P_gnss.

3.1 Travail à réaliser

Effectuez une étape de mise à jour de l'état du robot et de la cartographie à chaque fois qu'une mesure GNSS est disponible. Vous pouvez créer une fonction *"student_ update_ gnss.m"* qui réalisera cette mise à jour.

4 Annexes

En pratique, pour mettre en place un filtre de Kalman, il faut toujours réaliser deux étapes qui se répètent à chaque itération :

1. une phase de prédiction : $x_{k|k-1} = f(x_{k-1|k-1}, u_{k-1})$
2. une phase de mise à jour : $x_{k|k} = x_{k|k-1} + K(z_k - h(x_{k|k-1}))$

On peut représenter le principe sur la figure circulaire suivante :

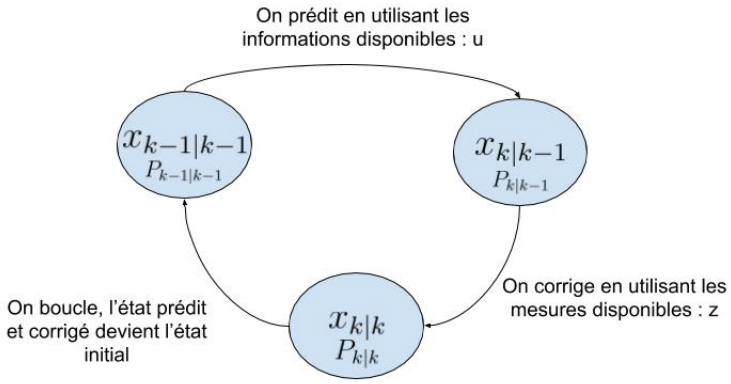
4.1 Prédiction

Cette épate consiste à prédire quelles seront les valeurs de l'état (et leurs covariances) après un temps donné dt en connaissant les informations sur le mouvement ou la commande actuelle.

$$x_{k|k-1} = f(x_{k-1|k-1}, u_{k-1})$$

4.1.1 Cas linéaire

$$x_{k|k-1} = Fx_{k-1|k-1} + Gu_{k-1}$$



Par exemple pour un robot dont l'état est $X = [x, y]^T$ qui est commandé en vitesse $u = \{V_x V_y\}$:

$$\begin{bmatrix} x \\ y \end{bmatrix}_{k|k-1} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{F} \begin{bmatrix} x \\ y \end{bmatrix}_{k-1|k-1} + \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{G} \begin{bmatrix} V_x \\ V_y \end{bmatrix}$$

De plus son incertitude ou erreur est représentée par sa covariance doit également être prédite. En effet, on connait l'erreur de l'état au temps k-1 ainsi que l'erreur sur la commande, l'état au temps k sera donc entaché de la propagation de ces deux erreurs. Cette propagation se fait comme suit :

$$P_{k|k-1} = F P_{k-1|k-1} F^T + G P_u G^T$$

4.1.2 Cas non linéaire

$$x_{k|k-1} = f(x_{k-1|k-1}, u_{k-1})$$

Soit une voiture autonome dont les paramètres d'état à estimer sont $X = [x, y, \theta]^T$. Ce véhicule est commandé en vitesse linéaire et angulaire $u = \{V_s, \omega_s\}$.

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k|k-1} = \begin{bmatrix} x + \underbrace{V_s dt \cos(\theta)}_{V_x} \\ y + \underbrace{V_s dt \sin(\theta)}_{V_y} \\ \theta + \omega_s dt \end{bmatrix}$$

Ici on ne peut pas décomposer sous forme linéaire. On a bien une fonction $f()$ qui relie l'état prédit avec l'état précédent et la commande. Pour propager l'incertitude, il faut utiliser les jacobiennes de la fonction $f()$ par rapport aux paramètres d'influence (ceux ayant une incertitude) :

$$P_{k|k-1} = J_{fX} P_{k-1|k-1} J_{fX}^T + J_{fu} P_u J_{fu}^T$$

Avec J_{fX} la jacobienne de $f()$ par rapport aux paramètres de X (de taille 3x3) et J_{fu} la jacobienne de $f()$ par rapport aux paramètres de u (de taille 3x2).

PS : si l'on souhaite également garder les vitesses V et ω dans l'état, X deviendrait alors :

$X = [x, y, \theta, V, \omega]^T$ et la prédiction serait : $\begin{bmatrix} x \\ y \\ \theta \\ V \\ \omega \end{bmatrix}_{k|k-1} = \begin{bmatrix} x + \underbrace{V_s dt \cos(\theta)}_{V_x} \\ y + \underbrace{V_s dt \sin(\theta)}_{V_y} \\ \theta + \omega_s dt \\ V_s \\ \omega_s \end{bmatrix}$

En gros, on pourrait écraser notre croyance actuelle de la vitesse par la mesure que l'on en fait (en supposant que la mesure soit plus précise que notre estimation actuelle ou si l'on ne sait pas du tout comment peut évoluer notre vitesse). Il faudrait alors calculer les jacobiennes correspondantes pour propager l'incertitude.

4.2 Mise à jour

Cette étape consiste à corriger la prédiction en obtenant des mesures issues de capteurs.

$$x_{k|k} = x_{k|k-1} + K \underbrace{(z_k - h(x_{k|k-1}))}_{Innovation}$$

L'idée consiste à dire quelle devrait être la mesure observée si l'on était vraiment à la position prédite $x_{k|k-1}$. Une fois cette "fausse" mesure estimée $\tilde{z}_k = h(x_{k|k-1})$, il suffit de la comparer avec la vraie mesure réalisée z_k. Cette différence est appelée innovation (quelle est la nouveauté apportée par la mesure).

Le Filtre de Kalman va alors venir ponderer par son gain K cet écart en prenant en compte les incertitudes respectives des différents paramètres.

Les étapes à réaliser sont donc :

1. Trouver la fonction d'observation $h()$ qui permet de prédire quelle mesure on aurait du avoir.
2. Calculer l'innovation $(z_k - h(x_{k|k-1}))$
3. Calculer la covariance de cette innovation : $S = J_{hX} P_{k|k-1} J_{hX}^T + P_z$
4. Calculer le gain de Kalman : $K = P_{k|k-1} J_{hX}^T S^{-1}$
5. Corriger l'état : $x_{k|k} = x_{k|k-1} + K(z_k - h(x_{k|k-1}))$
6. Calculer la covariance de l'état corrigé : $P_{k|k} = (I - K J_{hX}) P_{k|k-1}$

Avec J_{hX}, la jacobienne de la fonction d'observation par rapport aux variables de l'état X.

4.2.1 Cas linéaire

Dans le cas linéaire, $\tilde{z}_k = h(x_{k|k-1})$ se simplifie et la mesure estimée est directement une combinaison linéaire de l'état : $\tilde{z}_k = H x_{k|k-1}$. On remplace alors les J_{hX} précédent par H.

Dans le cas du robot d'état $X = [x, y]^T$ équipé d'un capteur de position fournissant $\{x_s, y_s\}$ on aurait alors :

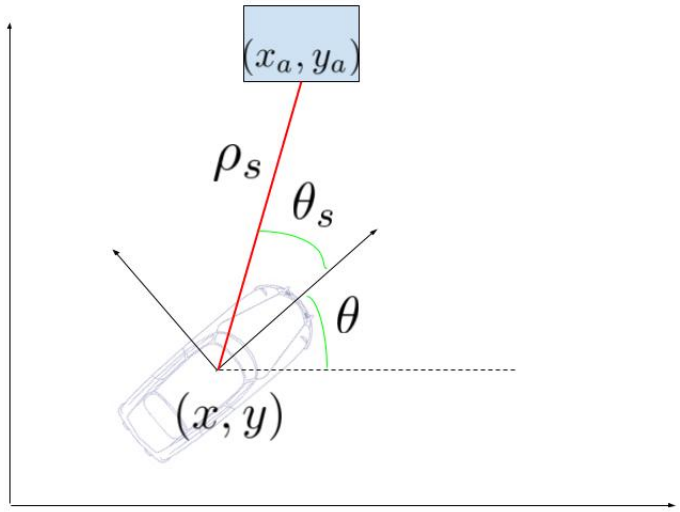
$$\tilde{z}_k = H x_{k|k-1} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{H} \begin{bmatrix} x \\ y \end{bmatrix}_{k|k-1}$$

On procèderait ainsi avec les équations de Kalman en utilisant cette valeur de H et cette prédiction $\tilde{z}_k$.

4.2.2 Cas non-linéaire

Très souvent, nos capteurs ne permettent pas d'observer directement un état du système mais une variable qui y est lié d'une façon non linéaire. C'est le cas lorsque l'on observe avec un laser des objets $[x_a y_a]^T$. En effet, le laser fournit des mesures en coordonnées polaires : $z = [\rho_s, \theta_s]^T$

Dans le cas de notre voiture autonome qui aurait une idée de la position de l'objet détecté avec $X = [x, y, \theta, x_a, y_a]^T$, comment prédire quelle devrait être la mesure polaire effectuée de cet objet situé en $[x_a y_a]^T$? Pour cela, il faut connaitre la distance entre l'objet et le véhicule ainsi que l'orientation du laser dans le repère véhicule (lui même peut tourner, or le laser est fixé sur le véhicule...)



$$\tilde{z}_k = h(x_{k|k-1}) = \begin{bmatrix} \tilde{\rho} \\ \tilde{\theta} \end{bmatrix}_k = \begin{bmatrix} \sqrt{(x-x_a)^2 + (y-y_a)^2} \\ \arctan\left(\frac{y_a - y}{x_a - x}\right) - \theta \end{bmatrix}$$

Même dans le cas simple d'une détection laser 2D, le modèle d'observation est non linéaire. En utilisant cette fonction $h()$, il suffit alors d'utiliser les équations de Kalman de la même manière que précédemment.

Les Jacobiennes d'un tel modèle utilisent les méthodes dérivatives classiques. Pour aide les Jacobienne de ce modèle sont données ci-dessous, la jacobienne de h est donc de taille 2xtaille(X) :

$$J_{hX} = \begin{bmatrix} \frac{\partial h}{\partial X_1} \cdots \frac{\partial h}{\partial X_n} \end{bmatrix}$$

$$J_{hX}(:, 1:5) = \begin{bmatrix} \frac{(x-x_a)}{\sqrt{(x-x_a)^2+(y-y_a)^2}} & \frac{(y-y_a)}{\sqrt{(x-x_a)^2+(y-y_a)^2}} & 0 & 0 & 0 \\ \frac{\frac{y_a-y}{(x_a-x)^2}}{1+u^2} & \frac{\frac{-1}{(x_a-x)}}{1+u^2} & -1 & 0 & 0 \end{bmatrix}$$

avec :

$$u = \frac{y_a - y}{x_a - x}$$

et :

$$J_{hX}(:, id_{ax} : id_{ay}) = \begin{bmatrix} \frac{-(x-x_a)}{\sqrt{(x-x_a)^2+(y-y_a)^2}} & \frac{-(y-y_a)}{\sqrt{(x-x_a)^2+(y-y_a)^2}} \\ \frac{\frac{-(y_a-y)}{(x_a-x)^2}}{1+u^2} & \frac{\frac{1}{(x_a-x)}}{1+u^2} \end{bmatrix}$$

Ailleurs :

$$J_{hX}(i, j) = 0$$

4.3 Cas du SLAM

Tout ce qui est dit jusqu'alors reste vrai dans le cadre du SLAM, la grosse différence vient du fait que la taille de l'état X et donc de sa covariance P varie dans le temps. De ce fait la taille des Jacobienne va également changer au fil du temps. Il n'y a aucun problème théorique, c'est en grande partie un problème de programamtion. Il faut donc bien réfléchir à comment utiliser tous ces éléments !