Estimer la trajectoire d’un robot buggy pour l'exploration d’un bâtiment inconnu à distance

Part 1 : Révision équation d'état, simulation avec Euler, MATLAB, coordonnées homogènes, dessin

1. Donner les équations d'état du buggy et les coder en MATLAB.

->

Détermination des entrées

En regardant comment le buggy est fait, on se rend compte qu’on a 2 entrées pour ce système : le moteur pour avancer et le servomoteur de direction des roues avant pour tourner (on va laisser de côté le servomoteur de direction des roues arrière, qui n’est pas indispensable). Nous noterons u(1) l’entrée qui contrôle la puissance moteur et u(2) l’entrée qui contrôle la direction.

Détermination des variables d’état

Si on veut dessiner le robot à un instant donné sur une carte, on sent bien qu'on a besoin :
_ De sa position x,y (que l'on ne mesure malheureusement pas directement)
_ De son cap theta (que l'on a assez directement avec la boussole)

Et si on veut avoir une idée d'où il va être prochainement :
_ Sa vitesse v (qui dépend assez directement de la puissance moteur)
_ L'angle du volant delta (que l'on commande assez directement)

Cependant, comme v et delta sont proportionnelles aux entrées, nous n’avons pas besoin de les inclure dans les variables d’état, elles seront juste utilisées comme variables intermédiaires car elles ont un sens physique utile notamment pour l’écriture de l’équation d’évolution.

Détermination de l’équation d’évolution

On peut s’inspirer très fortement du modèle de la voiture vu en cours (a priori vu plus en détails en 1A). Cependant, si on téléguide le buggy, on se rend compte que physiquement, la commande de puissance moteur peut être considérée en première approximation comme à peu près proportionnelle à la vitesse du buggy. Ceci est différent du modèle du cours, où c’était la dérivée de la vitesse (accélération de la voiture) qui était considérée proportionnelle à u(1). En effet, l’entrée u(1) de la voiture du cours était plutôt l’accélérateur. De même, ce n’est pas la dérivée de la direction des roues que l’on contrôle avec u(2), mais bien l’angle de la direction elle-même dans le cas de notre buggy. A priori, la voiture du cours contrôlait en vitesse un moteur relié au volant via son entrée u(2) (ceci peut s’expliquer par le fait qu’on doive faire plusieurs tours avec le volant de certaines voitures pour tourner à fond, on ne peut donc pas juste contrôler l’angle du volant). Il vaut mieux donc prendre ce type d’équation pour notre buggy :

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \alpha u_1 \cos(\beta u_2)\cos\theta \\ \alpha u_1 \cos(\beta u_2)\sin\theta \\ \frac{\alpha u_1 \sin(\beta u_2)}{L} \end{pmatrix}$$

Les coefficients alpha, beta, L seront à déterminer en fonction des caractéristiques physiques du buggy et sont tels que :

$v = \alpha u_1$

$\delta = \beta u_2$

Note : pour l'instant, on ne va trop réfléchir à l'équation d'observation.

2. Dessiner le buggy à un état donné.

3. Simuler le buggy avec Euler en train de faire un cercle (avec une commande constante simple).

->On peut par exemple mettre u(1) et u(2) à 0.5 pour que le robot ait une vitesse constante et un angle de roues constant.

4. S'inspirer de http://www.ensta-bretagne.fr/lebars/Share/Ecrazor_game.zip pour pouvoir utiliser les touches du clavier pour contrôler en direct le robot simulé.

->Remplacer f.m et draw.m et adapter les initialisations des variables, voir http://www.ensta-bretagne.fr/lebars/Share/buggy_simu.zip*...*

Part 2 : Comprendre ce qu'est un observateur d'état et pourquoi on en a souvent besoin

Exemple du robot buggy pour l'exploration de l'intérieur d'un bâtiment à distance : pour la localisation en intérieur, on connait :

- Les mesures directes : boussole électronique, par contre le GPS ne marche pas...
- Les commandes moteurs (puisque c'est nous qui les donnons).

Récupérer http://www.ensta-bretagne.fr/lebars/Share/buggy_real.zip et le tester pour contrôler le buggy virtuel/réel via MATLAB (lire ReadMe.txt pour faire les réglages nécessaires selon le buggy utilisé, bien vérifier et modifier si nécessaire les paramètres pour que le robot aille bien droit et dans la bonne direction).

1. Comment estimer très approximativement la trajectoire du buggy dans un premier temps ?

*->On va regarder quelle distance il parcourt en e.g. 10 s lorsque sa puissance moteur est à e.g. 25% pour trouver à quelle vitesse en m/s il avance et après on fera d=v*t selon le temps qu'il a avancé, et on utilisera sa boussole pour savoir à quel cap il était...*

Avec ça, on vient déjà de définir un observateur d'état, i.e. on est capable de calculer la position du robot (par rapport au point de départ) alors qu'on n'avait pas de capteur pour la mesurer directement !

*On a dû utiliser les capteurs (boussole) et les entrées envoyées au robot (puissance moteur), ainsi que des paramètres physiques que l'on a dû estimer par des tests (implicitement un coefficient alpha t.q. v=alpha*puissance moteur).*

2. Inversement, comment faire en sorte que le buggy suive une trajectoire simple prédéfinie, e.g une trajectoire formant un "L" de 3 m de côté ?

->En prenant en compte les mesures faites précédemment, on va dire au buggy de suivre e.g. le cap 0 (Nord) pendant 5 s, tourner de 90 deg (malheureusement, sur un buggy il faut avancer un peu pour pouvoir tourner donc on ne va pas être très précis), puis avancer encore 5 s.

Ceci constitue une combinaison entre :
_ Une commande en vitesse en boucle ouverte : on dit juste au robot d'avancer un certain temps, sans regarder la vraie vitesse ni combien il a réellement parcouru et donc sans pouvoir corriger
_ Et une commande en cap en boucle fermée : on a regardé la boussole pour pouvoir tourner de 90 deg et maintenir le cap

3. En considérant que les équations du robot simulé Part 1 sont réalistes, les réutiliser dans le code du buggy virtuel/réel pour dessiner sa position estimée dans MATLAB. On notera x_hat le vecteur d'état estimé et f_hat la fonction MATLAB qui contient les équations d'état estimées, pour faire la distinction avec l'état réel et les équations réelles du buggy que l'on ne peut pas connaître exactement. Cette méthode d'estimation s'appelle navigation à l'estime ou dead-reckoning.

4. On pourra ensuite considérer que l'angle de cap du robot est mesuré précisément par la boussole et donc modifier f_hat pour que cet angle soit comme une entrée.

->Voir http://www.ensta-bretagne.fr/lebars/Share/buggy_real_dead-reckoning.zip*...*
Grâce à un observateur d'état estimant la position à tout moment, on devrait pouvoir remplacer la commande en vitesse en boucle ouverte par une commande en position en boucle fermée, pour pouvoir suivre des trajectoires définies par des waypoints.

Part 3 : Vers le filtre de Kalman

Exemple du robot buggy pour l'exploration d'une zone extérieure à distance (récupérer http://www.ensta-bretagne.fr/lebars/Share/buggy_real_gps.zip) : cette fois-ci on a le GPS qui va nous donner des mesures directes de position, mais sa précision est de l'ordre de 5 m...

1. Réfléchir aux inconvénients des 2 méthodes d'estimation de position qu'on a maintenant à notre disposition.

->D'un côté le GPS a une précision de 5 m ce qui est assez grand w.r.t. la taille du robot (e.g. dans 5 m il y a assez de place pour que le robot fasse un cercle, alors que le GPS pourrait dire qu'il ne bouge pas…). D'un autre côté, plus le temps passe et plus le robot bouge, plus

les erreurs d'estimation de position avec la méthode « observateur utilisant les équations d'une simu d'Euler » vont s'accumuler et au bout d'un moment l'estimation de position ne sera plus réaliste…

2. Comment tirer le meilleur des 2 ?

->Il faut fusionner GPS et observateur précédent, une utilisant les 2 il y a de bonnes chances qu'on finisse par avoir une estimation de position qui combine les avantages des 2 et corrige un peu les inconvénients de chacun :
_ Moyenne entre les 2, c'est le principe de base du filtre de Kalman : il y a une prédiction (donnée par l'observateur d'état) et une correction (donnée par une mesure directe par GPS)!
_ Moyenne pondérée parce qu'on a plus confiance en le GPS que nos calculs...

Ce qu'on fait ici correspond en gros à ce qu'il y a dans des autopilotes type ArduPilot que l'on trouve sur des drones du commerce (e.g. quadrirotors pour la vidéo professionnelle...).

Ce pseudo filtre de Kalman non optimal et non formalisé est ici appliqué pour fusionner des infos de position, mais ça peut être utilisé pour la fusion d'autres types d'informations, e.g un module GPS utilise lui-même en interne un filtre de Kalman pour retrouver les signaux des satellites (de forme connue) dans le bruit radio ambiant...

L'utilisation d'un filtre de Kalman nous permettra par la suite de dessiner en plus les ellipsoïdes de confiance, qui nous donneront une idée assez claire de notre précision d'estimation, et ceci de manière optimale (minimisation des erreurs de modèle et de mesures au sens des moindres carrés) si on vérifie certaines hypothèses (bruits gaussiens sur les capteurs…)…