TD no2 : Commande d'un robot mobile

L'objectif de ce TD est d'implémenter et de tester des lois de commande en position et en orientation pour un robot mobile à roues différentielles.

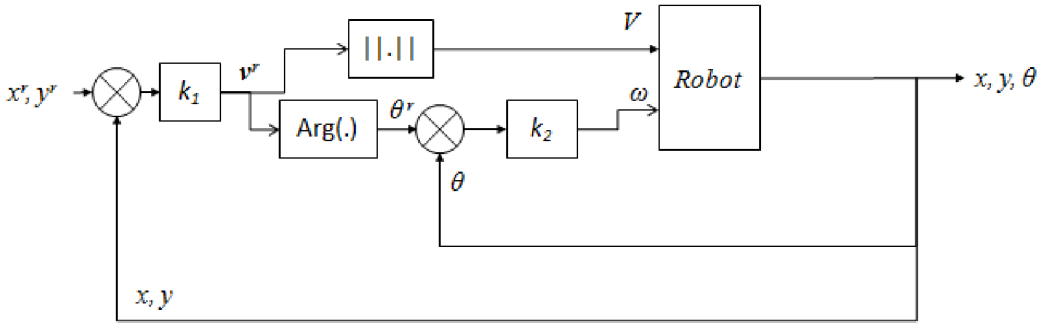
Le fichier Python *Robot.py* définit plusieurs classes d'objets qui seront nécessaires à l'implémentation :

Classe *Robot* : définit un objet de type robot par la donnée de son vecteur d'état (position (x,y) orientation θ) et de son vecteur de commande (module de la vitesse V, vitesse de rotation ω).

Classe *WPManager* : définit un objet permettant de gérer une liste de points de passage et de définir la position de référence (x^r, y^r) à atteindre par le robot.

Classe *RobotSimulation* : définit un objet permettant de stocker les informations liées à la gestion du temps lors d'une simulation (ensemble des dates de la simulation, pas de temps) et de stocker l'historique de l'état du robot, de sa commande, de la référence et des grandeurs de commande calculées. Cette classe définit également des fonctions d'affichage de ces historiques.

I. Commande en cascade pour rejoindre une position donnée



I.1. A partir du fichier Python *RobotNavWP.py*, compléter l'implémentation des boucles de commande en position et en orientation correspondant à la figure ci-dessus (correcteurs proportionnels pour les boucles de commandes en position et en orientation).

Fonctions python utiles : numpy.arctan2(x,y) numpy.sqrt(x)

I.2. Comment régler les gains des boucles de commande en position et en orientation ?

I.3. Valider votre implémentation et vos choix de réglages sur les trois scénarios suivants : $x_0 = 0$m, $y_0 = 0$m, $\theta_0 = 0$deg

Scénario1 : $x^r = 2$m, $y^r = 2$m Scénario2 : $x^r = 2$m, $y^r = 0$m Scénario3 : $x^r = 0$m, $y^r = 2$m

En déduire l'utilité des lignes de code suivantes qui suivent le calcul de l'angle de référence θ^r :

```
if (math.fabs(robot.theta-thetar)>math.pi):
        thetar = thetar + math.copysign(2*math.pi,robot.theta)
```

II. Navigation par points de passages

On souhaite maintenant que le robot suive une série de points de passage.

II.1) Quelle(s) structure(s) de données paraît(paraissent) adaptée(s) au stockage des points de passage ?

II.2) Complétez le fichier *RobotNavWP.py* pour implémenter le test de passage d'un point de passage à l'autre. Tester l'implémentation réalisée sur la succession de points de passage suivante :

No WP	1	2	3	4	5	6
x^r	2	2	-2	-2	2	0
y^r	2	-2	-2	2	2	0

II.3) Tester plusieurs valeurs pour le paramètre *epsilonWP*. Quelle influence peut-t-on observer sur la trajectoire réalisée par le véhicule ?

III. Génération de trajectoires et navigation par points de passages

On souhaite maintenant coupler la méthode de génération de trajectoires implémentée dans le TD no1 à la stratégie de navigation par points de passage de la partie II de ce TD.

III.1) Créez un fichier python *RobotNavWPAstar.py* et utilisez la carte issue du fichier *Map_20x10.py*. On souhaite générer une trajectoire sous forme de points de passages, entre les points de coordonnées (0,0) et celui de coordonnées (19,9). Utilisez la série de points générés et la stratégie de navigation par points de passage développée pour permettre au robot de se déplacer entre ces deux points dans la carte.

IV. Navigation par champs de potentiels

Le problème considéré dans cette partie est de permettre au robot de se rendre à un point donné tout en évitant des obstacles et sans avoir planifié au préalable de trajectoire de référence à suivre.

Le fichier Python *Potentials.py* définit plusieurs classes d'objets qui seront nécessaires à l'implémentation :

Classe *Goal* : définit un objet de type point de passage objectif par la donnée de ses coordonnées et du coefficient ka utilisé dans la définition du potentiel d'attraction associé (cf. plus bas)

Classe *Obstacle* : définit un objet de type obstacle par la données de ses coordonnées, de sa distance d'influence do (cf. plus bas) et des coefficients définissant les écarts types du potentiel de répulsion gaussien associé (cf. plus bas)

Potentiel d'attraction quadratique

IV.1) Dans la classe *Goal*, compléter les fonctions *evalAttractionPotential* et *evalAttractionForce* pour définir le potentiel et la force d'attraction définis par :

$$U_a = \frac{1}{2} k_a \left((x - x^r)^2 + (y - y^r)^2 \right) \qquad F_a = -\overrightarrow{\text{grad}} U_a$$

Vous pouvez utilisez les fonctions d'affichage définies dans la classe *Goal* pour visualiser le potentiel et la force d'attraction sur un exemple numérique de votre choix.

IV.2) Complétez le fichier *RobotNavPotFields.py* pour calculer la commande en position à partir de la force d'attraction ci-dessus. Testez votre implémentation avec comme point à rejoindre, le point de coordonnées (4,1) et comme position initiale pour le robot celle de votre choix.

Potentiel de répulsion Gaussien

IV.3) Dans la classe *Obstacle*, compléter les fonctions *evalRepulsionGaussianPotential* et *evalRepulsionGaussianForce* pour définir le potentiel et la force de répulsion définis par :

$$U_r = \begin{cases} \dfrac{1}{2\pi\sigma_x\sigma_y} \exp\left(-\dfrac{1}{2} \left\{ \left(\dfrac{x - x_o}{\sigma_x} \right)^2 + \left(\dfrac{y - y_o}{\sigma_y} \right)^2 \right\} \right) & \text{si} \quad d \le d_o \\ 0 \quad \text{sinon} \end{cases} \qquad F_r = -\overrightarrow{\text{grad}} U_r$$

où d_o modélise la zone d'influence de l'obstacle et d définit la distance entre le point (x, y) et le centre (x_o, y_o) de l'obstacle.
Vous pouvez utilisez les fonctions d'affichage définies dans la classe *Obstacle* pour visualiser le potentiel et la force d'attraction sur un exemple numérique de votre choix.

IV.4) Complétez le fichier *RobotNavPotFields.py* pour tenir compte de deux obstacles dans le calcul de la commande en position. Testez votre implémentation en sur les trois scénarios ci-dessous. Qu'observe-t-on pour chacun des scénarios en termes de comportement du robot ?

Etat initial du robot :
Scénario 1 : $x_0 = -4$, $y_0 = 1.2$, $\theta_0 = 0$
Scénario 2 : $x_0 = -4$, $y_0 = 0.8$, $\theta_0 = 0$
Scénario 3 : $x_0 = -4$, $y_0 = 1.0$, $\theta_0 = 0$

Point à atteindre : $x_r = 4$, $y_r = 1$, $k_a = 0.04$

Obstacles :

	x_o	y_o	d_o	σ_x	σ_y
Obstacle 1	-1.5	1	1	0.5	0.5

Obstacle 2	1.5	0	1	0.5	0.5

IV.5) A partir des observations faites à la question précédentes pour les trois scénarios, proposez une modification pour améliorer le comportement du robot. Vérifiez votre proposition en l'implémentant.