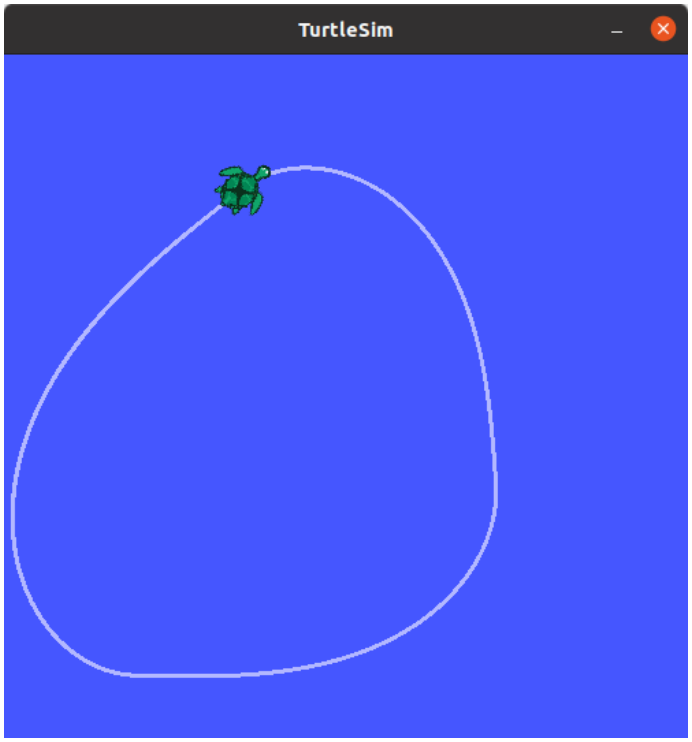

TP1 Note : Régulation

Partie 0 : création de package et projet git

1. Créer un projet sous github, que vous nommerez turtle_regulation_{nom1}_{nom2}
2. Dans votre catkin_ws dans le dossier src, faire un git clone de turtle_regulation_{nom1}_{nom2}
3. Créer un package avec pour nom turtle_regulation_{nom1}_{nom2} et pour dépendance rospy, roscpp, std_msgs, geometry_msgs et turtlesim
4. Faire un catkin build
5. Mettre à jour le répertoire github (git add . puis git commit -m " le message de commit" puis git push)

Partie 1 : Régulation en cap

Dans cette partie on propose de faire une régulation en cap.

1. Créer un nœud set_way_point.py dans turtle_regulation_{nom1}_{nom2}
2. Ce nœud doit souscrire à un topic « pose » de type turtlesim/Pose, lorsque l'on reçoit un message, une variable global (la pose de la tortue) doit être mise à jour.
3. Définir une variable waypoint avec pour coordonnées (7,7)

4. Pour calculer l'angle d'une droite passant par les points A (x_A, y_A)et B (x_B, y_B) on utilise la formule suivante : $\theta_{desired} = atan2(y_B - y_A, x_B - x_A)$

   Pour calculer l'angle désiré il faut calculer l'angle de la droite passant par la tortue et le waypoint.

   Calculer l'angle désiré.

5. Nous allons maintenant calculer la commande en cap du robot. On définit une erreur e par :

$$e = \arctan\left(\tan\left(\frac{\theta_{desired} - \theta}{2}\right)\right)$$

   Avec θ: l'angle du robot

   La commande u sera proportionnelle à l'erreur : $u = K_p * e$

   Publier un cmd_vel avec u la vitesse angulaire en z. Kp sera une constante, ce sera un paramètre privé du nœud.

6. Tester votre nœud avec différentes valeurs de Kp, dans le readme.md de votre répertoire git, préciser les comportements pour des Kp forts, des Kp faibles et le Kp que vous avez choisi.
7. Faire un commit de votre travail et mettre à jour le serveur github.

Partie 2 : Régulation en distance

1. Pour calculer une distance euclidienne entre deux points A et B, on utilise la formule suivante :

$$\sqrt{(y_B - y_A)^2 + (x_B - x_A)^2}$$

   Calculer la distance euclidienne entre le waypoint et la position de la tortue.
2. On définit l'erreur linéaire e_l entre deux points comme étant la distance entre ces deux points. La commande linéaire v sera proportionnelle à l'erreur linéaire, tel que $v = K_{pl} * e_l$, modifier la commande envoyée en ajoutant la composante vitesse linéaire. Kpl sera un paramètre privé de votre nœud.
3. Ajouter un seuil qu'on appellera distance_tolerance, quand la distance l'erreur linéaire est inférieur à distance_tolerance alors on ne publie plus sur cmd_vel. Distance_tolerance sera aussi un paramètre privé du nœud.
4. Ajouter un publisher qui publie sur le topic "is_moving" de type booléen. Lorsque l'erreur linéaire est supérieure à la distance_tolerance alors on publie True sinon False.
5. Tester votre nœud avec différentes valeurs de Kpl, dans le readme.md de votre répertoire git, préciser les comportements pour des Kpl forts, des Kpl faibles et le Kpl que vous avez choisi.
6. Faire un commit de votre travail et mettre à jour le serveur github.

Partie 3 : Création d'un service

Dans cette partie on va créer un service qui aura pour but de modifier le waypoint du nœud set_way_point.py

1. Créer un dossier srv dans votre package et ajouter un fichier waypoint.srv, ce srv file aura pour requête un std_msgs/Float32 x et un std_msgs/Float32 y et en réponse un std_msgs/Bool res.
2. Modifier les fichiers package.xml et CMakeList.txt pour pouvoir utiliser le srv file
3. Compiler, sourcer le setup.bash et vérifier avec rossrv list que votre srv est bien sur le système.
4. Dans set_way_point.py ajouter un service "set_waypoint_service" avec pour type way_point. Lorsque l'on appelle le service, la valeur de la variable waypoint de la question 3 de la partie 1 est modifiée.
5. Tester Votre service.
6. Faire un commit de votre travail et mettre à jour le serveur github.

Partie 4 : Création d'un client

1. Créer un client pour le service, ce client devra faire trois appels au service set_waypoint_service. Attention on doit envoyer une requête lorsque la tortue n'est pas en mouvement.
2. Faire un commit de votre travail et mettre à jour le serveur github.

Partie 5 : Création d'un launch file et rédaction du readme.md

1. Créer un launch file
2. Rédiger le README.md pour expliquer comment utiliser votre package.
3. Faire un commit de votre travail et mettre à jour le serveur github.