

## Chat history

New chat

Today

- Créer package et projet Git

Yesterday

- Revenus Spotify: 400M Streams
- Demande d'extension délai.
- Cisco Packet Tracer Network
- New chat
- Cisco Network Design Guide
- Network Topology and Components

Previous 7 Days

- New chat
- Homogenous vs Non-Homogenous
- New chat
- Python Arduino Button Monitoring
- Installer PyFirmata avec Arduino
- Installer Arduino sur Linux
- Lecture Bouton Arduino.
- Arduino Blink Script
- Cours Alimentation Île Maurice
- Création de package avec paramètres
- Install teleop\_twist\_keyboard
- Installer VMware.
- Explication ROS pour débutant
- New chat
- Énigme mathématique résolue.
- Création tableau étudiants.
- Syndrome Transfuseur-Transfusé: Information et Sensibilisation
- Conseils pour réussir.
- New chat

Previous 30 Days

- Mettre à jour clavier.
- Utilité d'un serveur DNS

Upgrade to PlusNEW



Hachim Hassani



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)

ChatGPT

Partie 0 : Création de package et projet git

1. Pour créer un projet sous GitHub, rendez-vous sur le site web de GitHub (<https://github.com>) et connectez-vous à votre compte.
2. Dans votre catkin\_ws, accédez au répertoire "src" en utilisant la commande suivante dans le terminal :

shellCopy code

```
cd catkin_ws/src
```

Ensuite, clonez le projet GitHub en utilisant la commande suivante :

shellCopy code

```
git clone <URL_du_projet_github>
```

Assurez-vous de remplacer "<URL\_du\_projet\_github>" par l'URL du projet que vous avez créé à l'étape 1.

3. Pour créer un package, utilisez la commande catkin\_create\_pkg avec les dépendances nécessaires. Exécutez la commande suivante dans le terminal :

shellCopy code

```
catkin_create_pkg turtle_regulation_<nom1>_<nom2> rospy roscpp std_msgs  
geometry_msgs turtlesim
```

Assurez-vous de remplacer "<nom1>" et "<nom2>" par les noms que vous avez choisis.

4. Ensuite, revenez au répertoire catkin\_ws en utilisant la commande suivante :

shellCopy code

```
cd ..
```

Puis, exécutez la commande catkin build pour compiler votre package :

shellCopy code

```
catkin build
```

5. Pour mettre à jour le répertoire GitHub avec les modifications effectuées, vous devez effectuer les étapes suivantes :

Tout d'abord, ajoutez les fichiers modifiés à l'index de Git en utilisant la commande suivante :

shellCopy code

```
git add .
```

Ensuite, effectuez un commit avec un message approprié :

shellCopy code

```
git commit -m "Le message de commit"
```

Enfin, poussez les modifications vers le dépôt GitHub :

shellCopy code

```
git push
```

Assurez-vous d'être positionné dans le répertoire "src/turtle\_regulation\_<nom1>\_<nom2>" lors de l'exécution de ces commandes.

Cela mettra à jour le répertoire GitHub avec vos modifications.



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 :  $\theta_{desired} = \text{atan2}(y_B - y_A, x_B - x_A)$  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 :  $e = \arctan(\tan(\theta_{desired} - \theta))$  Avec  $\theta$  : 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.  $K_p$  sera une constante, ce sera un paramètre privé du nœud. 6. Tester votre nœud avec différentes valeurs de  $K_p$ , dans le `readme.md` de votre répertoire git, préciser les comportements pour des  $K_p$  forts, des  $K_p$  faibles et le  $K_p$  que vous avez choisi. 7. Faire un commit de votre travail et mettre à jour le serveur github.

ChatGPT

Partie 1 : Régulation en cap

6. Pour créer un nouveau nœud appelé "`set_way_point.py`" dans votre package "`turtle_regulation_<nom1>_<nom2>`", vous pouvez utiliser la commande suivante :

shellCopy code

```
cd catkin_ws/src/turtle_regulation_<nom1>_<nom2>
touch set_way_point.py
```

Cela créera un nouveau fichier Python nommé "`set_way_point.py`" dans le répertoire de votre package.

7. Dans le fichier "set\_way\_point.py", vous devez souscrire au topic "pose" de type "turtlesim/Pose" pour mettre à jour une variable globale contenant la pose de la tortue. Voici un exemple de code pour cela :

pythonCopy code

```
#!/usr/bin/env python
```

```
import rospy
```

```
from turtlesim.msg import Pose
```

```
# Callback function to update the turtle's pose
```

```
def pose_callback(msg):
```

```
    global turtle_pose
```

```
    turtle_pose = msg
```

```
rospy.init_node('set_way_point')
```

```
rospy.Subscriber('pose', Pose, pose_callback)
```

```
# Global variable to store turtle pose
```

```
turtle_pose = Pose()
```

```
rospy.spin()
```

8. Définissez une variable "waypoint" avec les coordonnées (7, 7). Vous pouvez l'ajouter au début du fichier "set\_way\_point.py" :

pythonCopy code

```
waypoint = (7, 7)
```

9. Pour calculer l'angle de la droite passant par la tortue et le waypoint, vous pouvez utiliser la formule suivante :

pythonCopy code

```
import math
```

```
# Calculate desired angle
```

```
delta_x = waypoint[0] - turtle_pose.x
```

```
delta_y = waypoint[1] - turtle_pose.y
```

```
desired_angle = math.atan2(delta_y, delta_x)
```

Assurez-vous d'avoir importé le module math dans votre script.

10. Pour calculer la commande en cap du robot, vous pouvez utiliser la formule suivante :

pythonCopy code

```
# Calculate error
```

```
error = math.atan(math.tan(desired_angle - turtle_pose.theta))
```

```
# Define Kp as a private parameter (adjust the value as needed)
Kp = rospy.get_param('~Kp', 1.0)

# Calculate control input
u = Kp * error

# Publish cmd_vel with angular velocity
pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
cmd = Twist()
cmd.angular.z = u
pub.publish(cmd)
```

Dans ce code, nous utilisons la valeur de  $K_p$  récupérée à partir des paramètres privés du nœud à l'aide de la fonction `rospy.get_param()`. Assurez-vous d'avoir importé le type `Twist` du module `geometry_msgs.msg`.

11. Vous pouvez maintenant tester votre nœud en exécutant votre code et en ajustant différentes valeurs de  $K_p$ . Pour documenter les comportements observés, vous pouvez les ajouter dans le fichier "readme.md" de votre répertoire Git.
12. Une fois que vous êtes satisfait de votre travail, effectuez un commit des modifications apportées au fichier "set\_way\_point.py" et mettez à jour le dépôt GitHub en utilisant les commandes suivantes :

shellCopy code

```
git add set_way_point.py
git commit -m "Added set_way_point node"
git push
```

Assurez-vous d'exécuter ces commandes depuis le répertoire de votre package "turtle\_regulation\_<nom1>\_<nom2>".

comment faire la 6

ChatGPT

Pour effectuer la partie 6 où vous testez votre nœud avec différentes valeurs de  $K_p$  et documentez les comportements observés, vous pouvez suivre ces étapes :

13. Exécutez votre nœud en utilisant la commande suivante dans un terminal :

shellCopy code

```
roslaunch turtle_regulation_<nom1>_<nom2> set_way_point.py
```

Assurez-vous d'avoir lancé le nœud `turtlesim_node` pour pouvoir recevoir les mises à jour de la pose de la tortue.

14. Testez votre nœud en utilisant différentes valeurs de  $K_p$ . Vous pouvez ajuster la valeur de  $K_p$  en modifiant la ligne correspondante dans le fichier "set\_way\_point.py" ou en

utilisant des paramètres privés. Par exemple, vous pouvez exécuter votre nœud avec différentes valeurs de Kp en utilisant la commande suivante :

shellCopy code

```
roslaunch turtle_regulation_<nom1>_<nom2> set_way_point.py _Kp:=1.0
```

Cela définit la valeur de Kp à 1.0 pour ce test spécifique.

15. Observez le comportement de la tortue lorsque vous modifiez la valeur de Kp. Notez comment la tortue se déplace et se régule en cap.
16. Documentez les comportements observés dans le fichier "readme.md" de votre répertoire Git. Vous pouvez décrire les effets des valeurs de Kp élevées et faibles sur le mouvement de la tortue. Par exemple, vous pouvez noter si des valeurs élevées de Kp conduisent à une réaction plus rapide mais instable de la tortue, tandis que des valeurs faibles de Kp entraînent une réponse plus lente mais plus stable.

N'oubliez pas de sauvegarder les modifications du fichier "readme.md" et de les pousser vers le dépôt GitHub en utilisant les commandes suivantes :

shellCopy code

```
git add readme.md
git commit -m "Updated readme with observations"
git push
```

Assurez-vous d'être dans le répertoire racine de votre projet Git pour exécuter ces commandes.

Voici le code final pour le nœud "set\_way\_point.py" qui implémente la régulation en cap :

pythonCopy code

```
#!/usr/bin/env python

import rospy
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
import math

# Global variable to store turtle pose
turtle_pose = Pose()

# Callback function to update the turtle's pose
def pose_callback(msg):
    global turtle_pose
    turtle_pose = msg

# Calculate desired angle
def calculate_desired_angle(waypoint):
    delta_x = waypoint[0] - turtle_pose.x
    delta_y = waypoint[1] - turtle_pose.y
```

```

    desired_angle = math.atan2(delta_y, delta_x)
    return desired_angle

# Calculate control input
def calculate_control_input(desired_angle, Kp):
    error = math.atan(math.tan(desired_angle - turtle_pose.theta))
    u = Kp * error
    return u

if __name__ == '__main__':
    rospy.init_node('set_way_point')
    rospy.Subscriber('pose', Pose, pose_callback)

    # Define waypoint
    waypoint = (7, 7)

    # Define Kp as a private parameter (default value: 1.0)
    Kp = rospy.get_param('~Kp', 1.0)

    # Create publisher for cmd_vel
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)

    rate = rospy.Rate(10) # 10 Hz

    while not rospy.is_shutdown():
        # Calculate desired angle
        desired_angle = calculate_desired_angle(waypoint)

        # Calculate control input
        u = calculate_control_input(desired_angle, Kp)

        # Publish cmd_vel with angular velocity
        cmd = Twist()
        cmd.angular.z = u
        pub.publish(cmd)

        rate.sleep()

```

Assurez-vous d'avoir importé les modules nécessaires (rospy, turtlesim.msg, geometry\_msgs.msg et math) dans votre script.

Pour exécuter le nœud avec différentes valeurs de Kp, vous pouvez utiliser la commande suivante :

shellCopy code

```
roslaunch turtle_regulation_<nom1>_<nom2> set_way_point.py _Kp:=1.0
```

Cela définit la valeur de  $K_p$  à 1.0 pour cet exemple. Vous pouvez ajuster la valeur de  $K_p$  selon vos besoins.

N'oubliez pas de sauvegarder les modifications du fichier "readme.md" et de les pousser vers le dépôt GitHub après avoir effectué les tests et documenté les comportements observés.

Regenerate response

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. [C](#)