SEGUIDOR DE LINHA

Relembrando Conceitos:

Nó é um arquivo executável que usam ROS para se comunicar com outros nós. O ROS é projetado para ser modular em uma escala de granulação fina; um sistema de controle de robô geralmente compreende muitos nós. Por exemplo, um nó controla um localizador de faixa de laser, um nó controla os motores de roda, um nó realiza a localização, um nó executa o planejamento de caminho, um nó fornece uma visão gráfica do sistema e assim por diante. Um nó do ROS é gravado com o uso de uma biblioteca do cliente ROS, como roscpp ou rospy.

Os nós se comunicam uns com os outros, passando mensagens. Uma **mensagem** é simplesmente uma estrutura de dados, compreendendo campos digitados. Tipos primitivos padrão (inteiro, ponto flutuante, booleano, etc.) são suportados, assim como matrizes de tipos primitivos.

As mensagens são roteadas por meio de um sistema de transporte com semântica de publicação / assinatura. Um nó envia uma mensagem publicando-a em um determinado tópico. O **tópico** é um nome usado para identificar o conteúdo da mensagem. Um nó que está interessado em um determinado tipo de dados assinará o tópico apropriado. Pode haver vários editores e assinantes simultâneos para um único tópico, e um único nó pode publicar e / ou assinar vários tópicos. Em geral, os editores e assinantes não estão cientes da existência dos outros. A ideia é dissociar a produção de informação do seu consumo. Logicamente, pode-se pensar em um tópico como um barramento de mensagens fortemente tipado. Cada barramento tem um nome e qualquer um pode se conectar ao barramento para enviar ou receber mensagens, desde que sejam do tipo certo.

Criando um pacote:

1. Use os comandos:

```
#Entrando na pasta source onde ficam os pacotes

cd ~/catkin_ws/src

# seguidor_linha é o nome do futuro pacote e o restante são as
dependências de mensagem

catkin_create_pkg seguidor_linha std_msgs rospy roscpp
cd ~/catkin_ws
catkin build
. ~/catkin_ws/devel/setup.bash
```

2. Criando um Nó em Python:

Para criar um arquivo executável, use os comandos:

```
roscd seguidor_linha
```

```
mkdir scripts
cd scripts
gedit sensor_luz.py  #sensor_luz é um nome para o nó
# salvar o arquivo sensor_luz.py antes do próximo comando
chmod +x sensor_luz.py   #faz com que o nó seja um arquivo executável
cd ~/catkin_ws
catkin build
```

Obs: Arquivos em python necessitam de ser compilados apenas na hora da criação. Depois, ao modificar é apenas necessário salvar o código.

3. Criando o código para o seguidor de linha:

Abra o arquivo sensor_luz.py (ou o arquivo que desejou criar).

No Wiki ROS possui vários exemplos explicados de como criar um simples código em python.

Link: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

Obs: Primeiramente tente fazer o seguidor de linha sozinho a partir do tutorial e exemplo do ROS Wiki.

Exemplo de código do seguidor de linha:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
from nxt_msgs.msg import Light
from geometry_msgs.msg import Twist
# geometry_msgs/Twist
# nxt_msgs/Light

light_l = 0.0
light_r = 0.0

light_tresh = 0.0

def callback_light_l(data):
    global light_l

    light_l = data.intensity     #rostopic echo
```

```
    #rospy.loginfo("left light: %s", light_l)

def callback_light_r(data):
    global light_r

    light_r = data.intensity
    #rospy.loginfo("right light: %s", light_r)

def listener():
    rospy.init_node('light_control', anonymous=True)
    pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
    rospy.Subscriber("/light_l", Light, callback_light_l)
    rospy.Subscriber("/light_r", Light, callback_light_r)

    # INSERT CALIBRATION ROUTINE HERE
    light_tresh = 400.0

    # spin() simply keeps python from exiting until this node is
stopped
    # rospy.spin()

    vel_msg = Twist()
    vel_msg.linear.x = 0
    vel_msg.linear.y = 0
    vel_msg.linear.z = 0
    vel_msg.angular.x = 0
    vel_msg.angular.y = 0
    vel_msg.angular.z = 0

    rate = rospy.Rate(10)
    while not rospy.is_shutdown():

        if light_l > light_tresh and light_r > light_tresh:
            rospy.loginfo("front")
            vel_msg.linear.x = 0.04
            vel_msg.angular.z = 0
```

```
        elif light_l < light_tresh:
            rospy.loginfo("counter clockwise")
            vel_msg.linear.x = 0
            vel_msg.angular.z = 0.5
        elif light_r < light_tresh:
            rospy.loginfo("clockwise")
            vel_msg.linear.x = 0
            vel_msg.angular.z = -0.5
        else:
            rospy.loginfo("stop")
            vel_msg.linear.x = 0
            vel_msg.angular.z = 0

        pub.publish(vel_msg)
        rate.sleep()

if __name__ == '__main__':
    listener()
```

a. Algumas dicas:

```
from nxt_msgs.msg import Light
from geometry_msgs.msg import Twist
```

Obtido no comando *rostopic type /cmd_vel* e *rostopic type /light*

```
light_l = data.intensity
```

rostopic echo light_l

```
while not rospy.is_shutdown():

        if light_l > light_tresh and light_r > light_tresh:
            rospy.loginfo("front")
            vel_msg.linear.x = 0.04
            vel_msg.angular.z = 0
        elif light_l < light_tresh:
            rospy.loginfo("counter clockwise")
```

```
            vel_msg.linear.x = 0
            vel_msg.angular.z = 0.5
        elif light_r < light_tresh:
            rospy.loginfo("clockwise")
            vel_msg.linear.x = 0
            vel_msg.angular.z = -0.5
        else:
            rospy.loginfo("stop")
            vel_msg.linear.x = 0
            vel_msg.angular.z = 0

        pub.publish(vel_msg)
        rate.sleep()
```

Loop do código. Será executado infinitas vezes até que desligue o código em python.

```
#rospy.loginfo("right light: %s", light_r)
```

rospy.loginfo serve para mostrar dados e mensagens no terminal de comando.

```
    # INSERT CALIBRATION ROUTINE HERE
    light_tresh = 400.0
```

O light_tresh é ultilizado como um tipo de limite de luz para que o comando seja executado. Ele deve ser obtido pelo *rostopic echo* dos dois sensores analisando quanto que o sensor da de valor na transição do branco para a faixa preta, ou seja, se a transição apresenta 400 de intensidade, um bom valor a ser utilizado no light_tresh seria aproximadamente 390.