ROS - Parte 2

Programação Robótica
Universidade Federal de Pernambuco
Adrien Durand-Petiteville
adrien.durandpetiteville@ufpe.br

Topics

- Nós se comunicam, trocando informações e dados.
- A maneira mais comum de fazer isso é através de tópicos (topics).
- **Um tópico é um nome para um fluxo de mensagens com um tipo definido.**
- Por exemplo, os dados de uma câmera podem ser enviados sobre um tópico chamado *image_topics*, com um tipo de mensagem *Image*.

publish/subscribe

- Os tópicos implementam um mecanismo de comunicação de publicação / assinatura (publish/subscribe).
- Antes de os nós começarem a transmitir dados sobre tópicos, eles devem primeiro anunciar, ou *advertise*, o nome do tópico e os tipos de mensagens que serão enviadas.
- Em seguida, eles podem começar a enviar, ou *publish*, os dados reais sobre o tópico.
- Os nós que desejam receber mensagens em um tópico podem se inscrever, ou *subscribe*, nesse tópico, solicitando o `roscore`.
- Após a assinatura, todas as mensagens no tópico são entregues no nó que fez a solicitação.

Publicando em um tópico - 1

```
#! /usr/bin/python

import rospy
# Importação da biblioteca de mensagens padrões
from std_msgs.msg import Int32

# Criação do nó com o nome simple_publisher
rospy.init_node('simple_publisher')

# Criação do publisher no topic counter de um mensagem de tipo Int32
pub = rospy.Publisher('counter', Int32, queue_size=1)

rate = rospy.Rate(2)

count = 0

while not rospy.is_shutdown():
    # Publicação da mensagem no topico
    pub.publish(count)

    count += 1
    rate.sleep()
```

Publicando em um tópico - 3

```
from std_msgs.msg import Int32
```

- Importa a definição da mensagem que vamos enviar sobre o tópico.
- Aqui, usamos um número inteiro de 32 bits, definido no pacote de mensagens padrão do ROS, std_msgs.
- Para que a importação funcione conforme o esperado, precisamos importar do `<nome do pacote>.msg`, pois é aqui que as definições do pacote são armazenadas.
- Como estamos usando uma mensagem de outro pacote, precisamos informar o sistema de criação do ROS sobre isso adicionando uma dependência ao nosso arquivo package.xml.

```
<depend package="std_msgs"/>
```

- Sem essa dependência, o ROS não saberá onde encontrar a definição da mensagem.

Publicando em um tópico - 4

```
pub = rospy.Publisher('counter', Int32)
```

- Anuncia o nó com um publicador.
- Fornece um nome ao tópico (counter) e especifica o tipo de mensagem que será enviada por ele (Int32).
- Neste ponto, o tópico é anunciado e está disponível para outros nós se inscreverem.

Publicando em um tópico - 5

```
count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

- Primeiro, definimos a taxa, em hertz, na qual queremos publicar.
- A função `is_shutdown()` retornará True se o está pronto para ser desligado e False caso contrário, para que possamos usá-lo para determinar se é hora de sair do loop `while`.
- Dentro do loop `while`, publicamos o valor atual do contador, aumentamos seu valor em 1 e depois dormimos um pouco.
- A chamada para `rate.sleep()` durará o suficiente para garantir que executemos o corpo do loop `while` em aproximadamente 2 Hz.

Verificar se tudo funciona como esperado

```
rostopic -h
```

```
rostopic list
```

```
rosrun package file.py
```

```
rostopic list
```

```
rostopic echo topic -n 5
```

```
rostopic hz topic
```

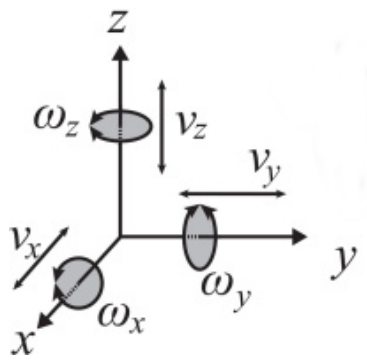
```
rostopic info topic
```

```
rostopic find std_msgs/Int32
```

Tipo de mensagem

ROS type	Serialization	C++ type	Python type	Notes
bool	Unsigned 8-bit integer	uint8_t	bool	
int8	Signed 8-bit integer	int8_t	int	
uint8	Unsigned 8-bit integer	uint8_t	int	uint8[] is treated as a string in Python
int16	Signed 16-bit integer	int16_t	int	
uint16	Unsigned 16-bit integer	uint16_t	int	
int32	Signed 32-bit integer	int32_t	int	
uint32	Unsigned 32-bit integer	uint32_t	int	
int64	Signed 64-bit integer	int64_t	long	
uint64	Unsigned 64-bit integer	uint64_t	long	
float32	32-bit IEEE float	float	float	
float64	64-bit IEEE float	double	float	
string	ASCII string	std::string	string	ROS does not support Unicode strings; use a UTF-8 encoding
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time	duration

A mensagem **TWIST**



- Para controlar um corpo com 6 graus de liberdade usamos a mensagem padrão **TWIST** `http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Twist.html`
- **TWIST** é uma estrutura com 6 campos:
  - v_x: TWIST.linear.x
  - v_y: TWIST.linear.y
  - v_z: TWIST.linear.z
  - w_x: TWIST.angular.x
  - w_y: TWIST.angular.y
  - w_z: TWIST.angular.z

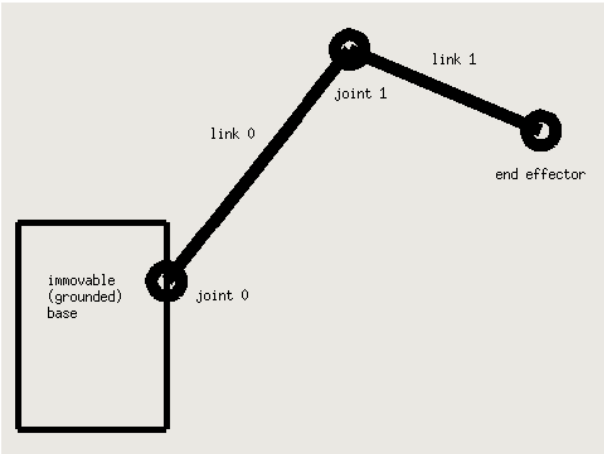
Mensagem para controlar um robô TurtleBot

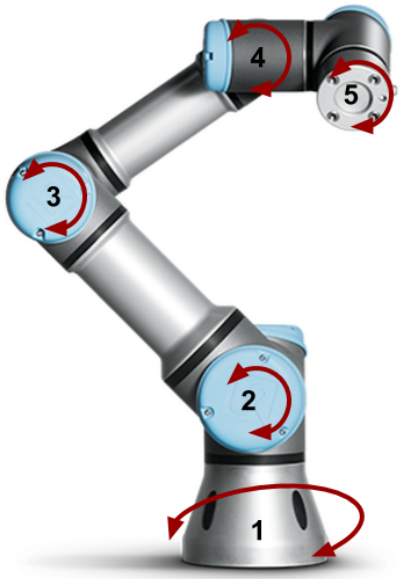
```
#! /usr/bin/python

import rospy
from std_msgs.msg import Int32
# Importação da biblioteca de mensagens de geometria
from geometry_msgs.msg import Twist

rospy.init_node('simple_publisher')

# Criação do publisher no topic /turtle/cmd_vel de um mensagem de tipo Twist
pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=1)

cmd = Twist()

rate = rospy.Rate(1)

count = 0
while not rospy.is_shutdown():

    if count % 2 == 0:
        cmd.linear.x = 1
        cmd.angular.z = 0
    else:
        cmd.linear.x = 0
        cmd.angular.z = 1

    pub.publish(cmd)
    count += 1
    rate.sleep()
```

Mensagem para controlar um robô TIAGo

```python
#! /usr/bin/python3

import rospy
from std_msgs.msg import Int32
from geometry_msgs.msg import Twist


rospy.init_node('simple_publisher')

# Criação do publisher no topic /mobile_base_controller/cmd_vel de um mensagem de tipo Twist
pub = rospy.Publisher('/mobile_base_controller/cmd_vel', Twist, queue_size=1)

cmd = Twist()

rate = rospy.Rate(1)

count = 0
while not rospy.is_shutdown():

    if count % 2 == 0:
        cmd.linear.x = 1
        cmd.angular.z = 0
    else:
        cmd.linear.x = 0
        cmd.angular.z = 1

    pub.publish(cmd)
    count += 1
    rate.sleep()
```

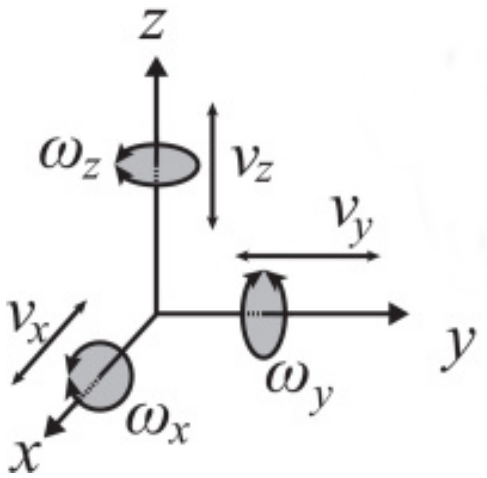
Manipuladores





- Manipuladores robóticos são um conjunto de juntas mantidas juntas por uma estrutura de algum tipo.
  - As juntas rotativas giram em torno de um eixo de rotação.
  - As juntas prismáticas movem-se linearmente ao longo de um eixo de movimento.
- Um link é uma seção de um braço de robô conectada por uma junta.

A mensagem **JointTrajectory**



- Para controlar um conjunto de juntas usamos a mensagem padrão **JointTrajectory**
  `http://docs.ros.org/en/melodic/api/trajectory_msgs/html/msg/JointTrajectory.html`
- **JointTrajectory** é uma estrutura com 3 campos:
  - `std_msgs/Header` **header**
  - `string[]` **joint_names**
  - `trajectory_msgs/JointTrajectoryPoint[]` **points**
- O nome das juntas deve corresponder ao nome no controlador (usar rostopic list & echo para encontrar)

A mensagem **std_msgs/Header**

- Para controlar um conjunto de juntas usamos a mensagem padrão **Header**
  `http://docs.ros.org/en/melodic/api/std_msgs/html/msg/Header.html`
- **Header** é uma estrutura com 3 campos:
  - `uint32` **seq**
  - `time` **stamp**
  - `string` **frame_id**

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.sec: seconds (stamp_secs) since epoch (in Python the variable is called 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (in Python the variable is called 'nsecs')
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
string frame_id
```

A mensagem **trajectory_msgs/JointTrajectoryPoint[]**

- Para criar uma trajetória usamos a mensagem padrão **JointTrajectoryPoint[]**
  `http://docs.ros.org/en/melodic/api/trajectory_msgs/html/msg/JointTrajectoryPoint.html`
- **JointTrajectoryPoint[]** é uma estrutura com 5 campos:
  - `float64[]` **positions**
  - `float64[]` **velocities**
  - `float64[]` **accelerations**
  - `float64[]` **effort**
  - `duration` **time_from_start**
- Apenas um dos campos pode ser usado por vez (**positions**, **velocities**, **acceleration** ou **effort**).
- O campo **time_from_start** é obrigatório.

Mensagem para controlar um robô TIAGo - 1

```
#! /usr/bin/python

import rospy
from std_msgs.msg import Int32
from trajectory_msgs.msg import JointTrajectory
# Importação da biblioteca de mensagens de trajetoria
from trajectory_msgs.msg import JointTrajectoryPoint

rospy.init_node('move_arm')

# Criação do publisher no topic /arm_controller/command
# de um mensagem de tipo JointTrajectory
pub = rospy.Publisher('/arm_controller/command', JointTrajectory, queue_size=1)

cmd = JointTrajectory()
cmd.joint_names.append("arm_1_joint")
cmd.joint_names.append("arm_2_joint")
cmd.joint_names.append("arm_3_joint")
cmd.joint_names.append("arm_4_joint")
cmd.joint_names.append("arm_5_joint")
cmd.joint_names.append("arm_6_joint")
cmd.joint_names.append("arm_7_joint")
```

Mensagem para controlar um robô TIAGo - 2

```
point = JointTrajectoryPoint()
point.positions = [0] * 7
point.time_from_start = rospy.Duration(1)

cmd.points.append(point)

rate = rospy.Rate(1)

angle = 0.1

while not rospy.is_shutdown():

    cmd.points[0].positions[1] = angle
    cmd.points[0].time_from_start = rospy.Duration(1)

    pub.publish(cmd)
    angle += 0.1
    rate.sleep()
```