
ROS

Universidade Federal de Santa Maria

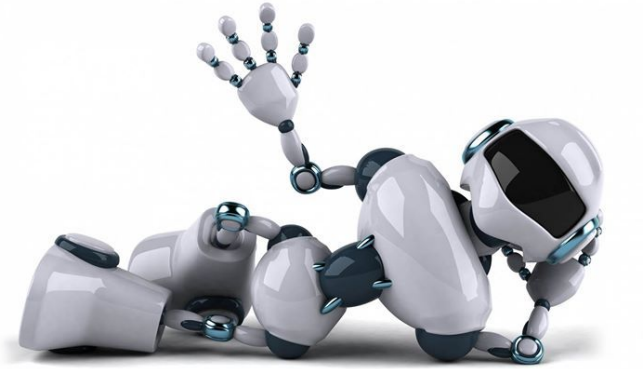
Introdução ROS

Robot Operating System

Fabricio, David, Julio

Sumário

- O que é ROS?
- Instalação/Configuração do ambiente ROS
- Criação do Workspace/Introdução ao sistema de arquivos
- Criando Pacote ROS
- Entendendo ROS nodes
- Entendendo ROS topics
- Entendendo nodes e topics com a Tartaruguinha
- Um simples Publisher/Subscriber em python
- Exemplo prático dynamixel



O que é ROS?

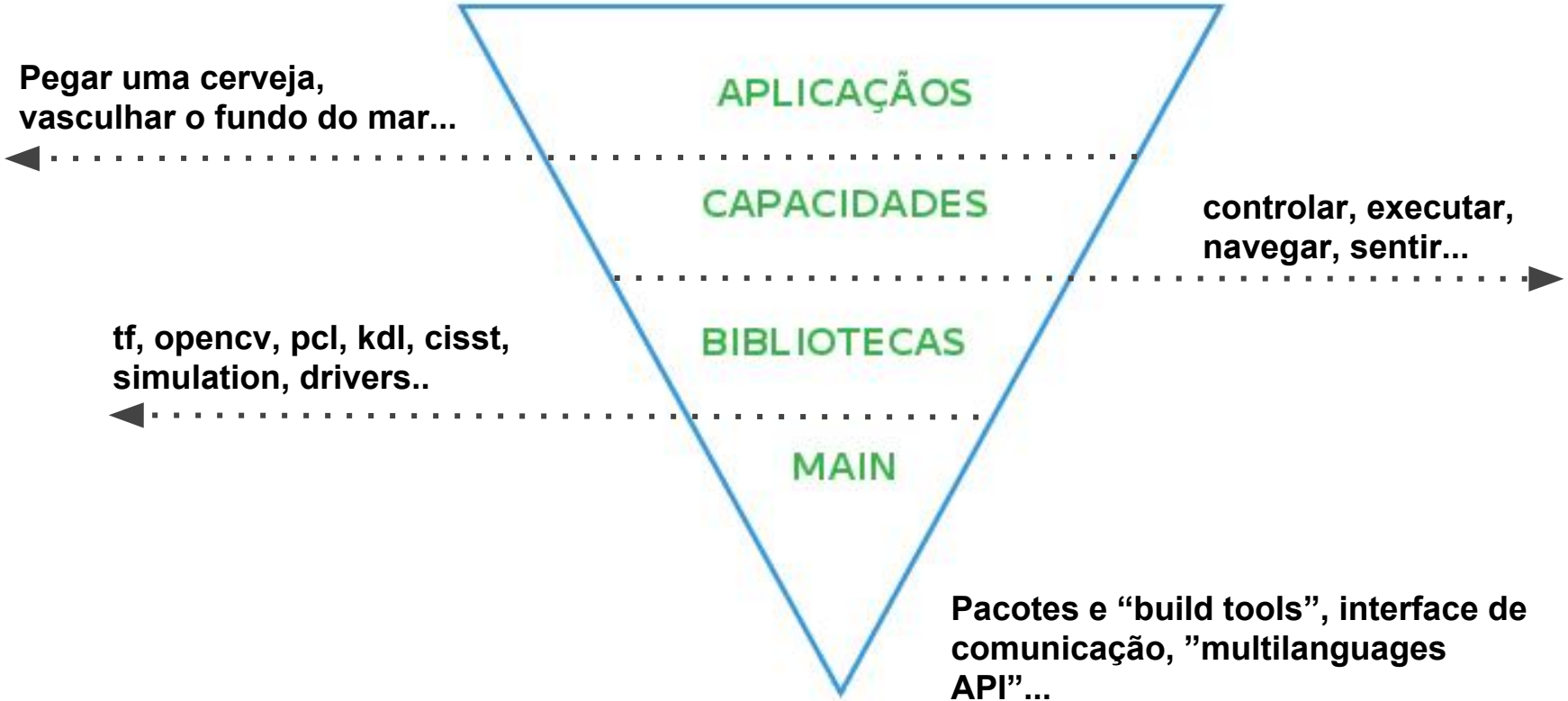
- É um “meta” sistema operacional para robôs
- Um conjunto de pacotes e ferramentas para desenvolvimento de softwares escaláveis multiplataforma/multilinguagem .
- Uma arquitetura distribuída para comunicação entre processos e equipamentos.
- Um arquitetura multilinguagem

“With the intent to enable researchers to rapidly develop new robotic systems without having to “reinvent the wheel” through use of standard tools and interfaces”

O que ROS não é?

- Um sistema operacional
- Uma linguagem de programação
- Uma IDE
- Um RTOS

O que podemos esperar do ROS?



NODES
(Nós)

O QUE SÃO OS NODES (NÓS)?

- Um nó para o ROS é onde se processa informações.
- É um executável que fica dentro de um pacote do ROS

- Nós se conectam em um grafo e se comunicam utilizando:
  - tópicos em *streaming;*
  - serviços de RPC;
  - Servidores de parâmetros.

- Geralmente vários nós existem em um projeto.

- Eles adicionam uma “tolerância” a falhas no projeto porque os erros, *bugs* e *crashes* que ocorrem, ficam restritos a cada nó.

O QUE SÃO OS NODES (NÓS)?

- Para criar um Nó no ROS é necessário utilizar uma das bibliotecas do ROS, como o ROSCP ou o ROSPY para ele se comunicar com outros Nós.

- Os nós podem publicar ou se inscrever em tópicos além de poderem prover ou utilizar algum serviço

TOPICS
(Tópicos)

O QUE SÃO TÓPICOS?

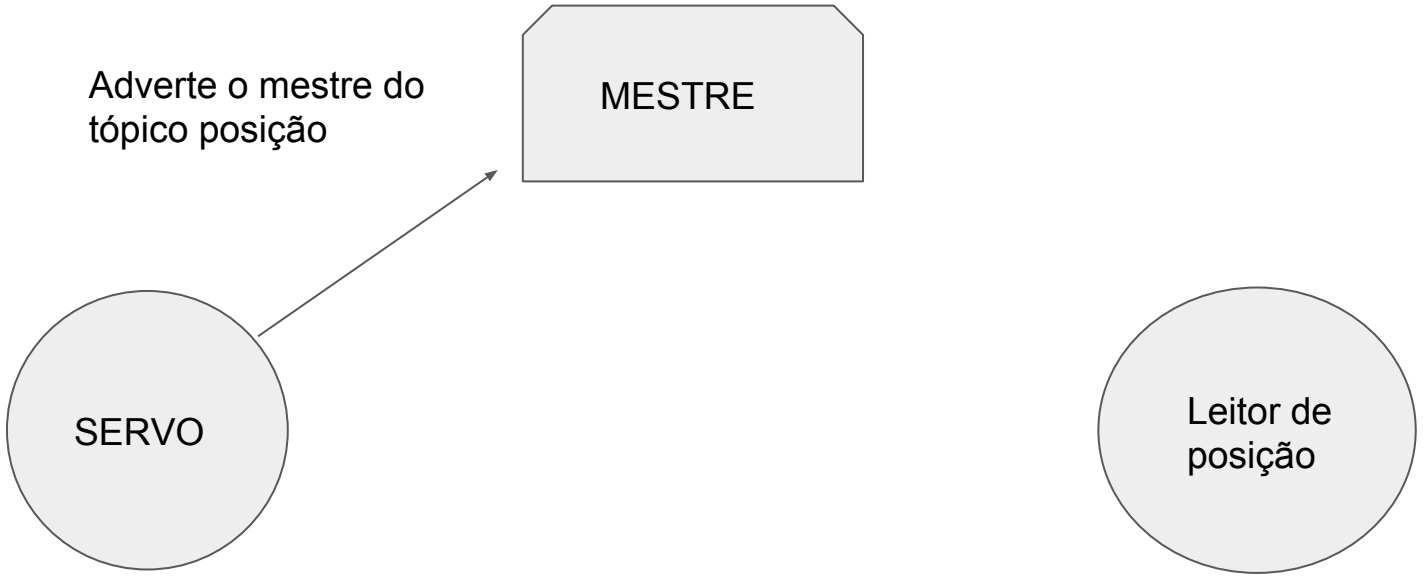
- Tópico no ROS é que conecta os nodos certos um no outro, como assim?

- Existem diferentes tópicos no ROS, e estes são definidos pelo tipo de mensagem que compartilham, MASOQ?!

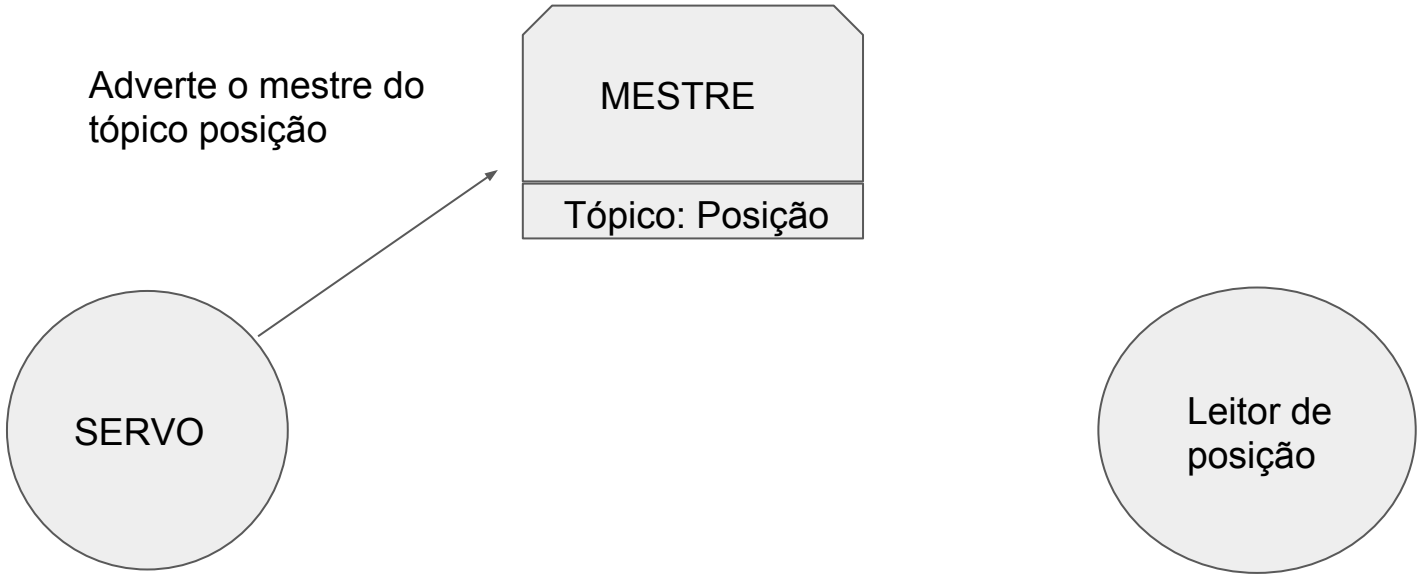
- Assim:
  - Algum nó começa um tópico mostrando para o “nó mestre” o que está transmitindo
  - Outro nó que só escuta pede para o “nó mestre” algum tópico que tenha o tipo de mensagem que este consegue processar…

Desenhando fica melhor!

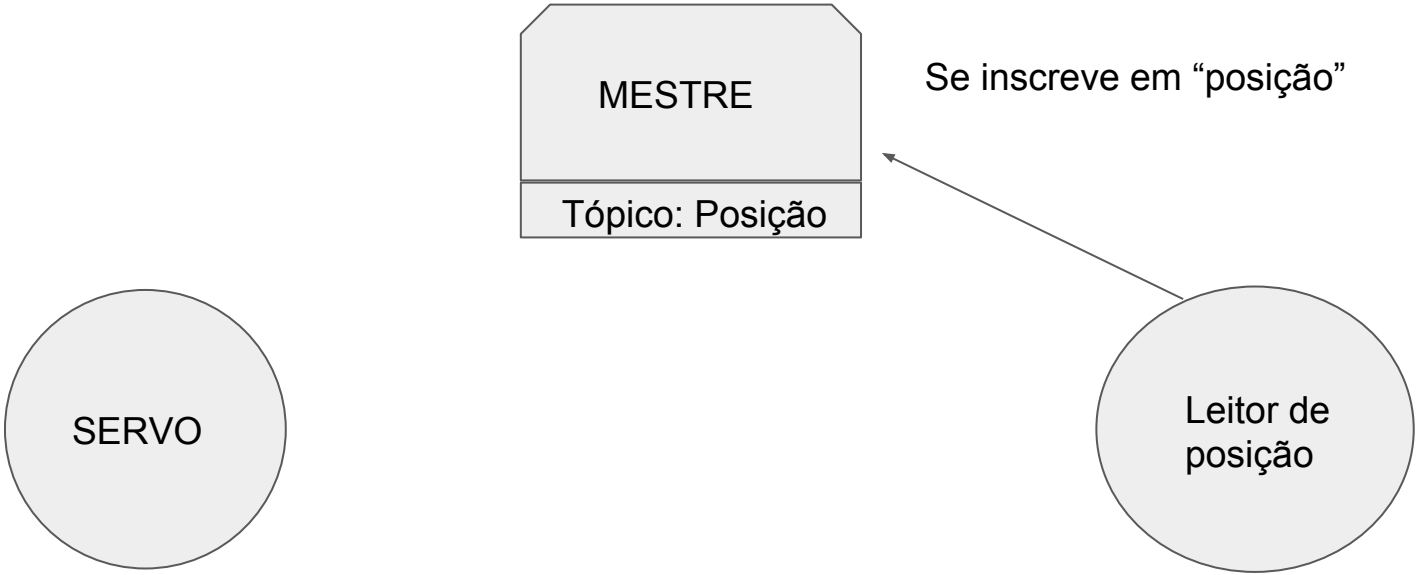
NÓS e TÓPICOS



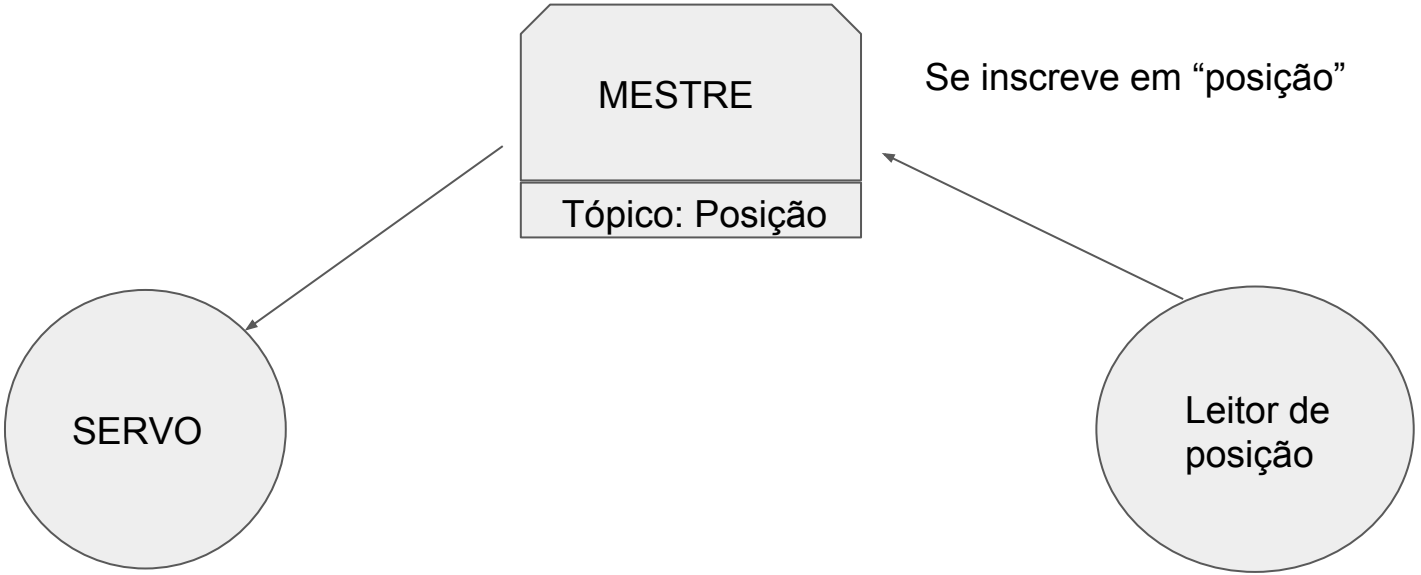
NÓS e TÓPICOS



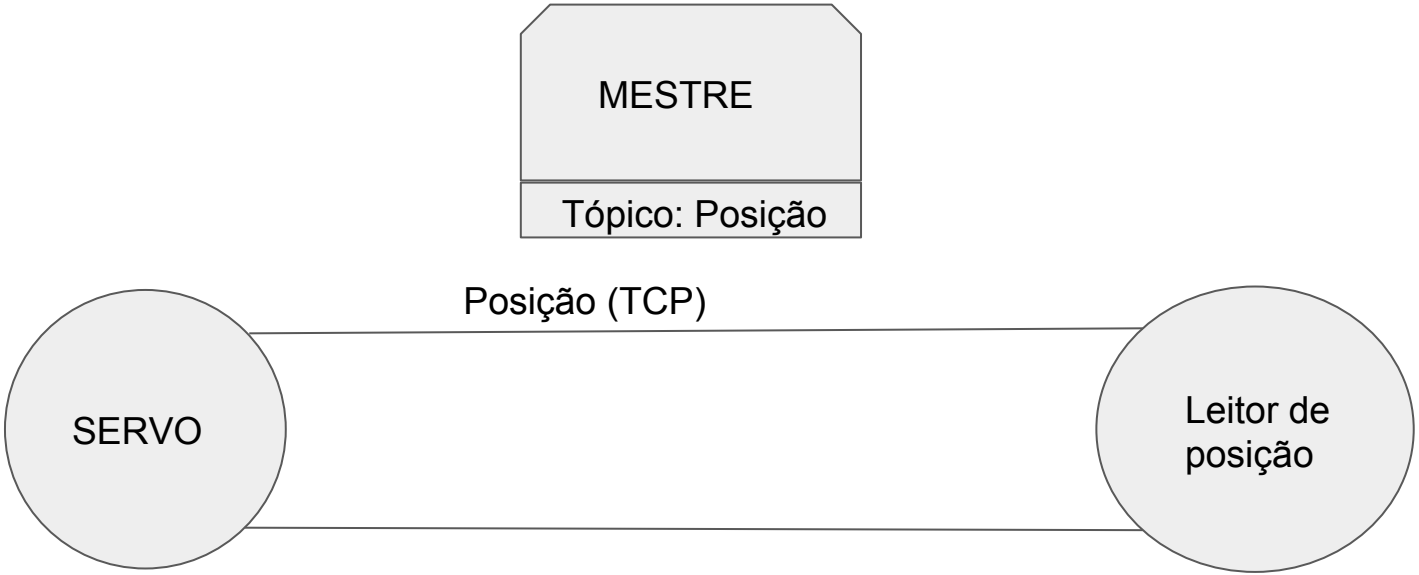
NÓS e TÓPICOS



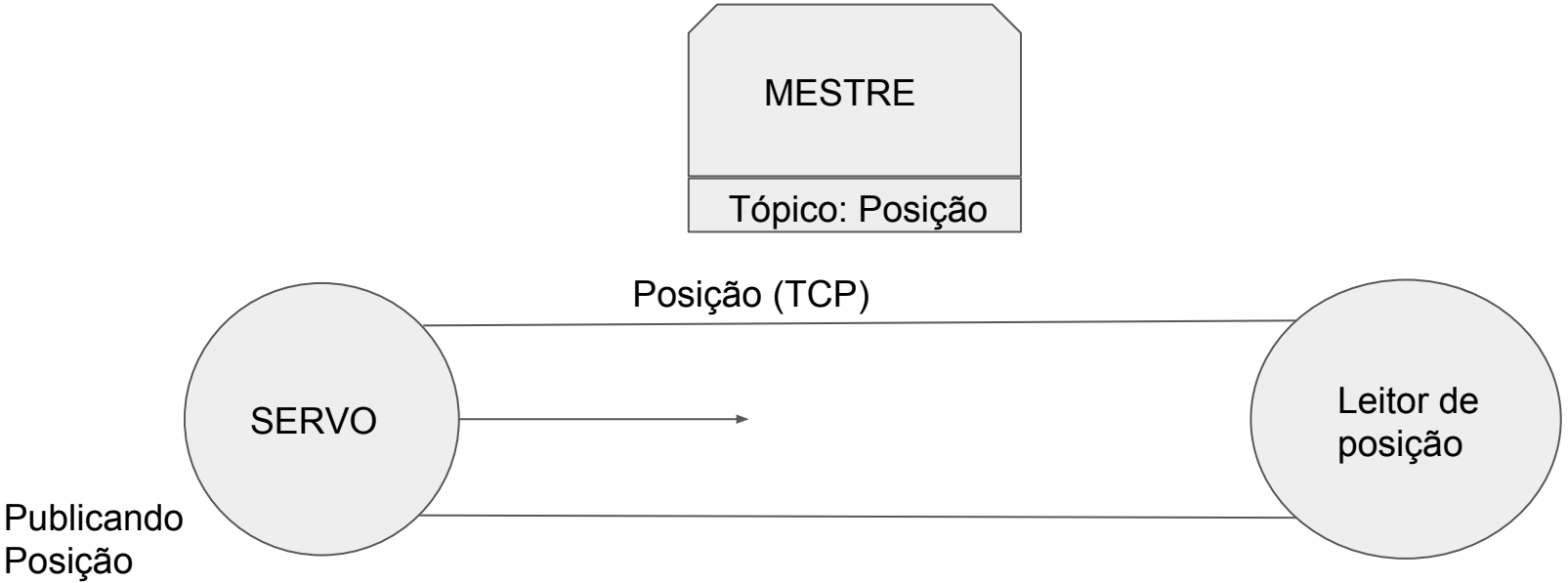
NÓS e TÓPICOS



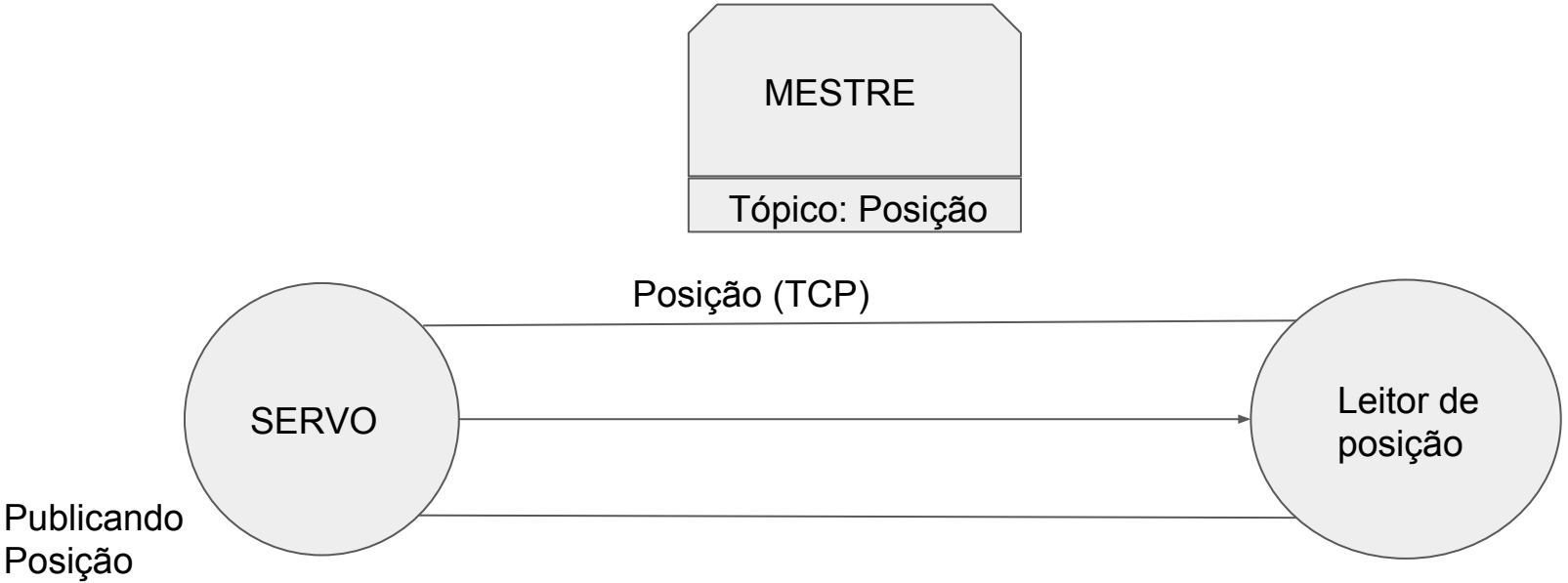
NÓS e TÓPICOS



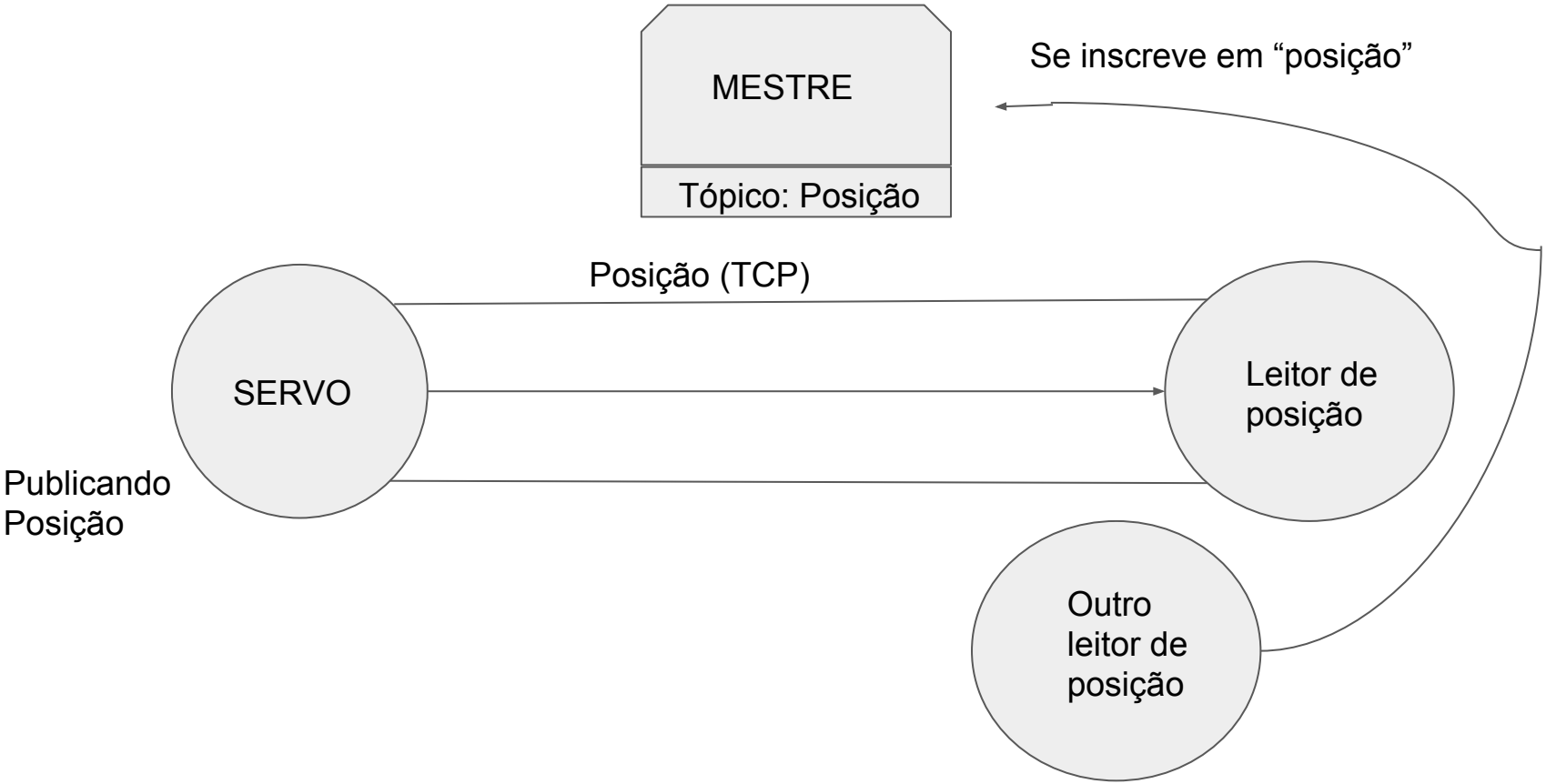
NÓS e TÓPICOS



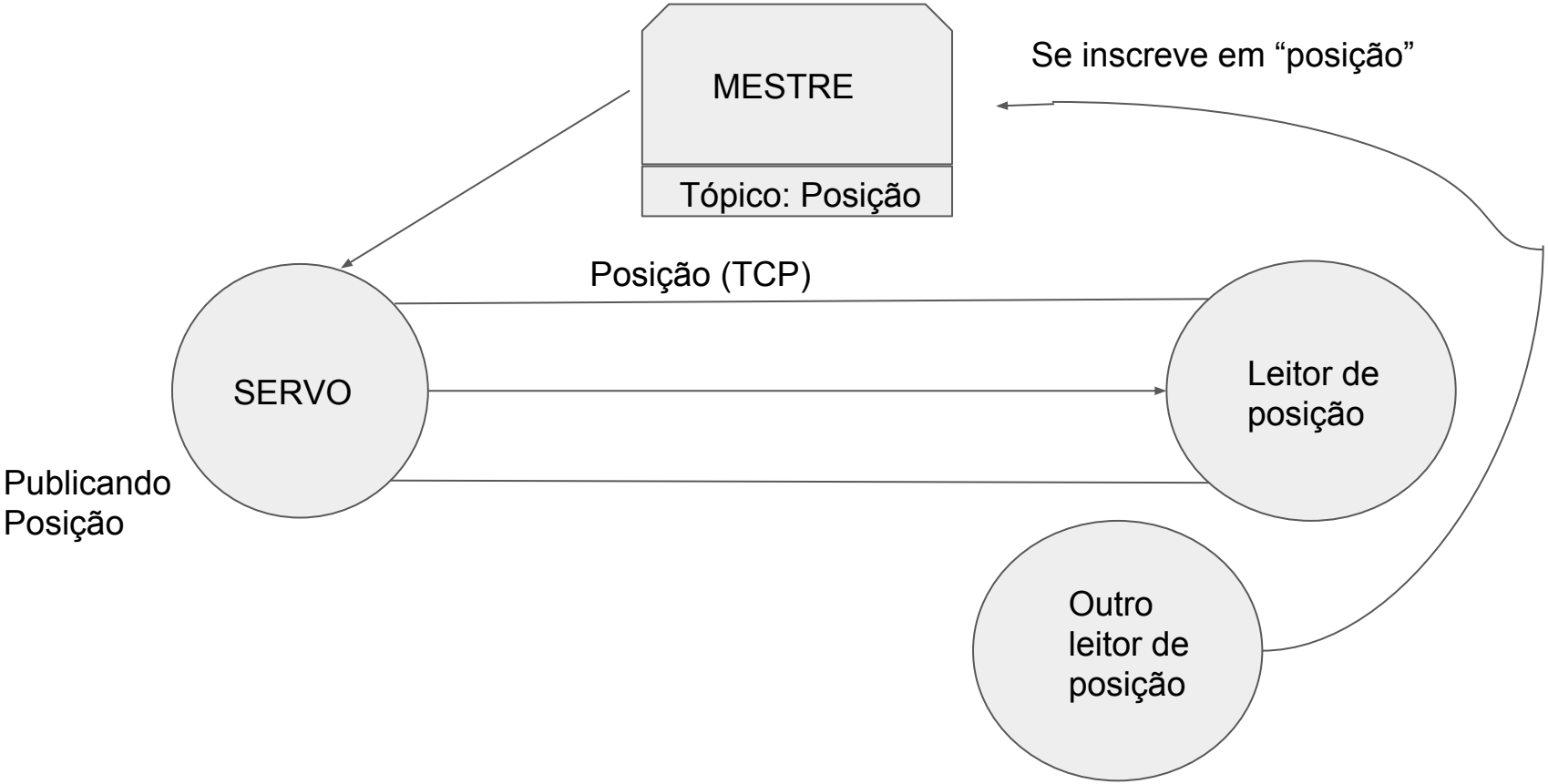
NÓS e TÓPICOS



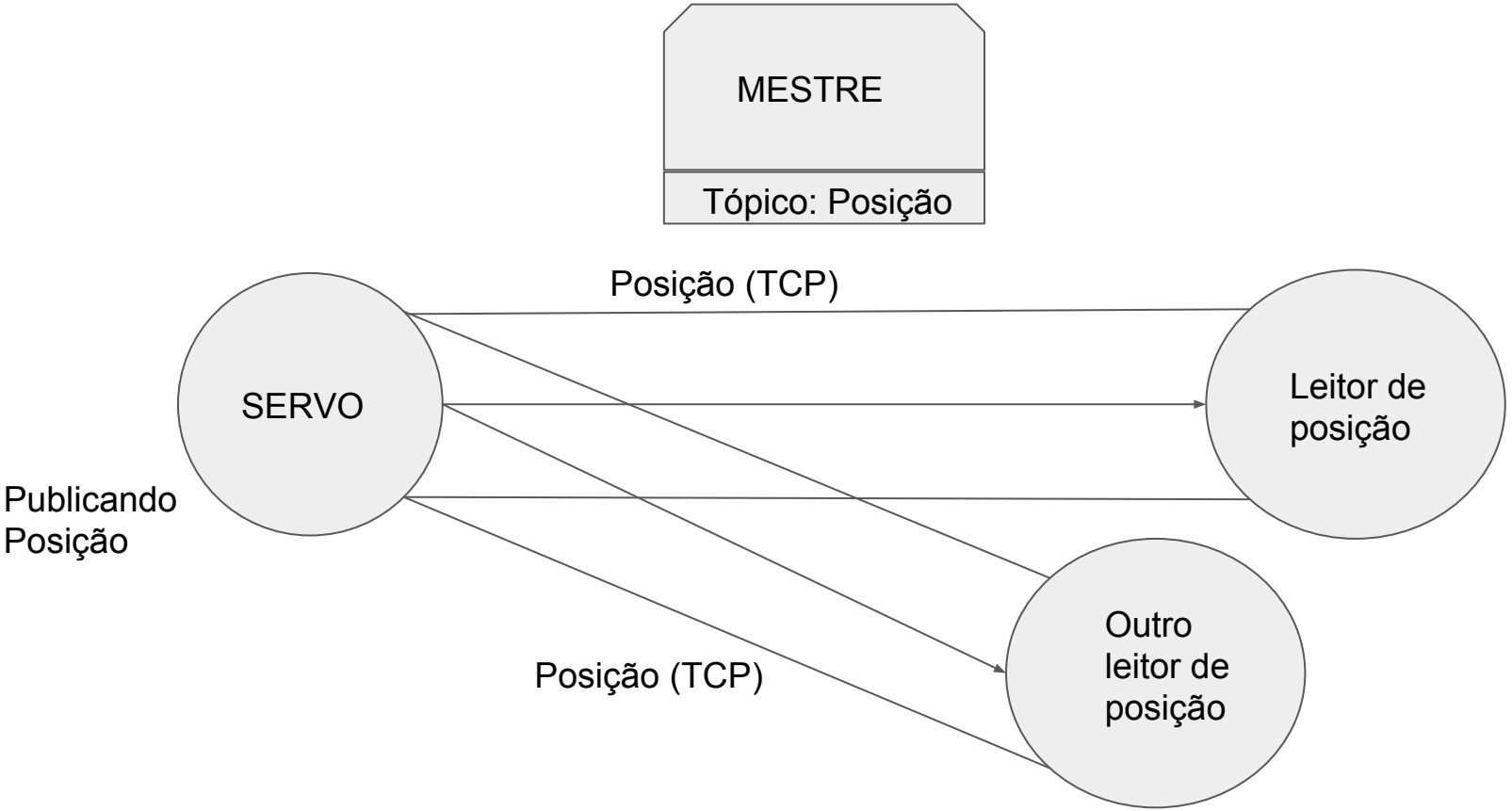
NÓS e TÓPICOS



NÓS e TÓPICOS

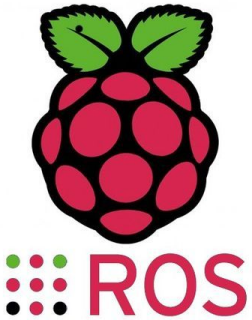


NÓS e TÓPICOS



Instalando ROS

- Plataformas disponíveis: Ubuntu(x86, x64, armf), são suportados, os demais estão em fase de desenvolvimento.
- Em nossa instalação utilizaremos uma VM
  - Xubuntu (ubuntu trusty 14.04.4 x64)
  - Usuário/Senha: unisctuto/123



Instalando ROS

- Modificando sources.list para adicionar a fonte de onde os pacotes do ROS serão obtidos

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

- Adiciona uma chave GPG, indicando ao apt-get que a assinatura digital do repositório é confiável.

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 0xB01FA116
```

- Instalando

```
sudo apt-get update
sudo apt-get install ros-jade-desktop-full
```

Instalando ROS

Adicionando variáveis de ambiente ao nosso bashrc

```
echo "source /opt/ros/jade/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Inicializando rosdep: ferramenta de gerenciamento de dependencias especifico para ROS

```
sudo rosdep init
rosdep update
```

Catkin Workspace ROS

Catking introduz o conceito de workspace onde podemos fazer “build” de múltiplos pacotes independentes na mesma execução. Um pacote catking é equivalente a um pacote Cmake.

O que é um pacote? O objetivo da criação de pacotes é agrupar funcionalidades uteis de uma forma intuitiva e fácil de usar.

O maior beneficio da utilização de pacotes é a possibilidade de reusar partes de de um software.

Estrutura de um Catkin Workspace ROS

```
catkin_ws/                -- WORKSPACE
  src/                    -- SOURCE SPACE
    ...
  build/                  -- BUILD SPACE
  devel/                  -- DEVEL SPACE
    setup.bash           \
    setup.sh             |-- Environment setup files
    setup.zsh            /
    etc/                  -- Generated configuration files
    include/              -- Generated header files
    lib/                  -- Generated libraries and other artifacts
      package_1/
        bin/
        etc/
        include/
        lib/
        share/
        ...
      package_n/
        bin/
        etc/
        include/
        lib/
        share/
    share/                -- Generated architecture independent artifacts
    ...
```

Criando um Workspace

- Criação das pastas e inicialização do workspace

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

- Mesmo que o espaço de trabalho esteja vazio (não existam pacotes no diretório 'src', simplesmente um simples link para uma CMakeLists.txt) você ainda pode dar build no espaço de trabalho

```
$ cd ~/catkin_ws/
$ catkin_make
```

- Adicionamos nosso workspace à nossas variáveis de ambientes.

```
$ source devel/setup.bash
```

Criando um pacote

- Para criação de pacotes utilizamos o seguinte comando

```
catkin_create_pkg <nome_pacote> [depend1] [depend2] [depend3]
```

```
$ cd ~/catkin_ws/src

$ catkin_create_pkg tutorial_unisc std_msgs rospy roscpp

$ catkin_make
```

Tutorial 1: Publicar e ouvir tópicos em python

```
$ roscd tutorial_unisc
```

```
$ mkdir scripts
$ cd scripts
```

Criaremos dois arquivos dentro desta pasta talker.py, listener.py:

```
nano talker.py
```

```
.
```

```
.
```

```
.
```

```
nano listener.py
```

Publicando uma mensagem em um tópico

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def talker():
    #Aqui declaramos que nosso no publicara a mensagem no topico1, string sera o tipo de mensagem
    pub = rospy.Publisher('topico1', String, queue_size=10)
    #A proxima linha diz ao rospy o nome do no que criaremos,
    #neste caso o no estara fazendo um brodcast de uma mensagem no topico1
    rospy.init_node('node_talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz e a frequencia de broadcast
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        #publica a mensagem no topico
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Escutando a mensagem de um tópico

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():
    #iniciamos um novo no que servirá para ouvir o topico1
    rospy.init_node('listener', anonymous=True)
    #Selecionamos o topico que queremos ouvir
    rospy.Subscriber("topico1", String, callback)
    # spin() mantem o codigo rodando enquando o no existir
    rospy.spin()

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

Turtle Sim

Preparação:

```
$ sudo apt-get install ros-<distro>-ros-tutorials
```

(no nosso caso substituimos distro pela nossa versão do ROS que é a _____)

Em seguida vamos dar o camando

```
$ roscore
```

Turtle Sim

Existe algum nó ativo no momento? Para ver isso usamos o comando

```
$ rosnode list
```

o único nó sera o /rosout

O que ele faz? Para saber isso pedimos informações com

```
$rosnode info /rosout
```

Node [/rosout]

Publications:

* /rosout_agg [rosgraph_msgs/Log]

Subscriptions:

* /rosout [unknown type]

Services:

* /rosout/set_logger_level

* /rosout/get_loggers

contacting node http://machine_name:54614/ ...

Pid: 5092

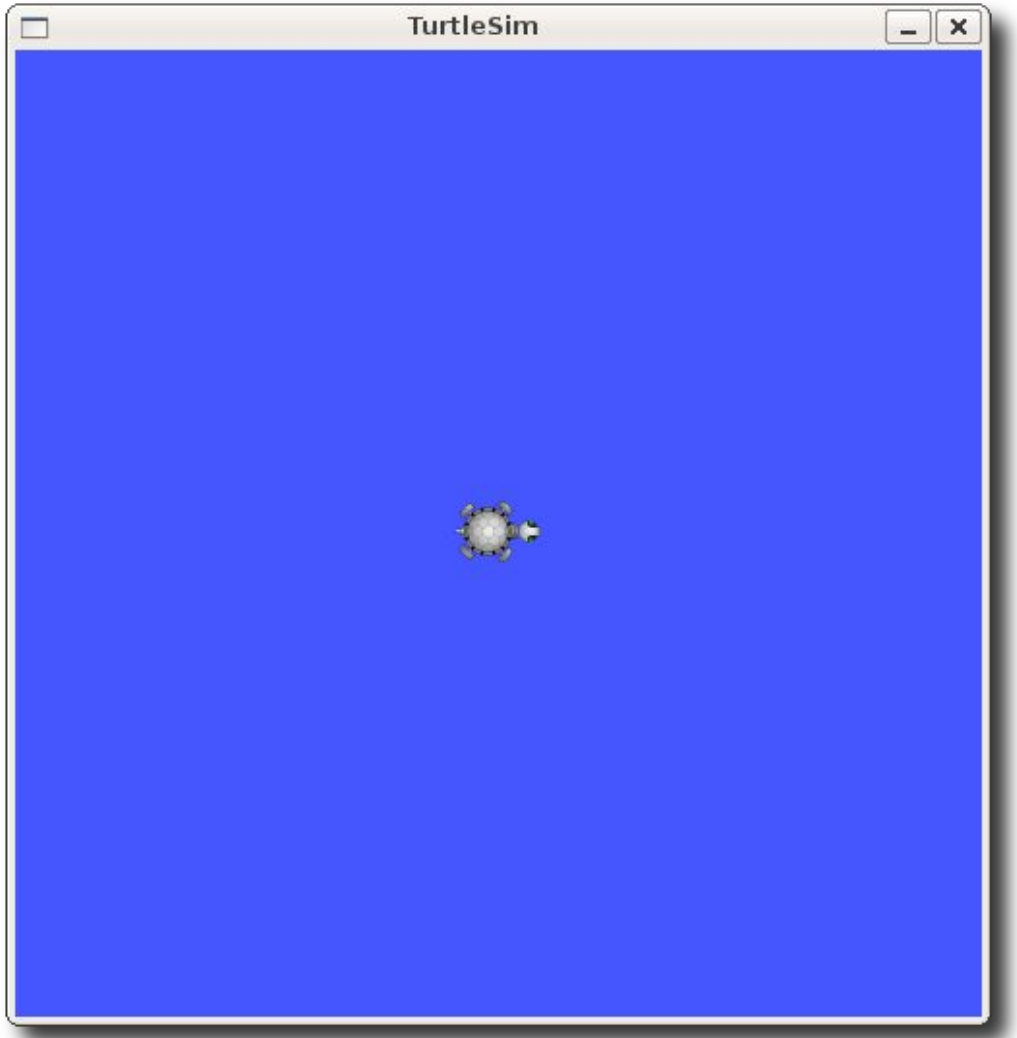
Turtle Sim

Como iniciar um novo nó?

```
$ rosrun [nome_pacote] [nome_nó]
```

Em um outro terminal vamos chamar este pacote e este nó

```
$ rosrun turtlesim turtlesim_node
```



Se eu utilizar o rosnode list agora?

```
$ rosnode list
```

Olha só o outro nó ali!

/rosout

/turtlesim

Turtle Sim

O que vimos até aqui:

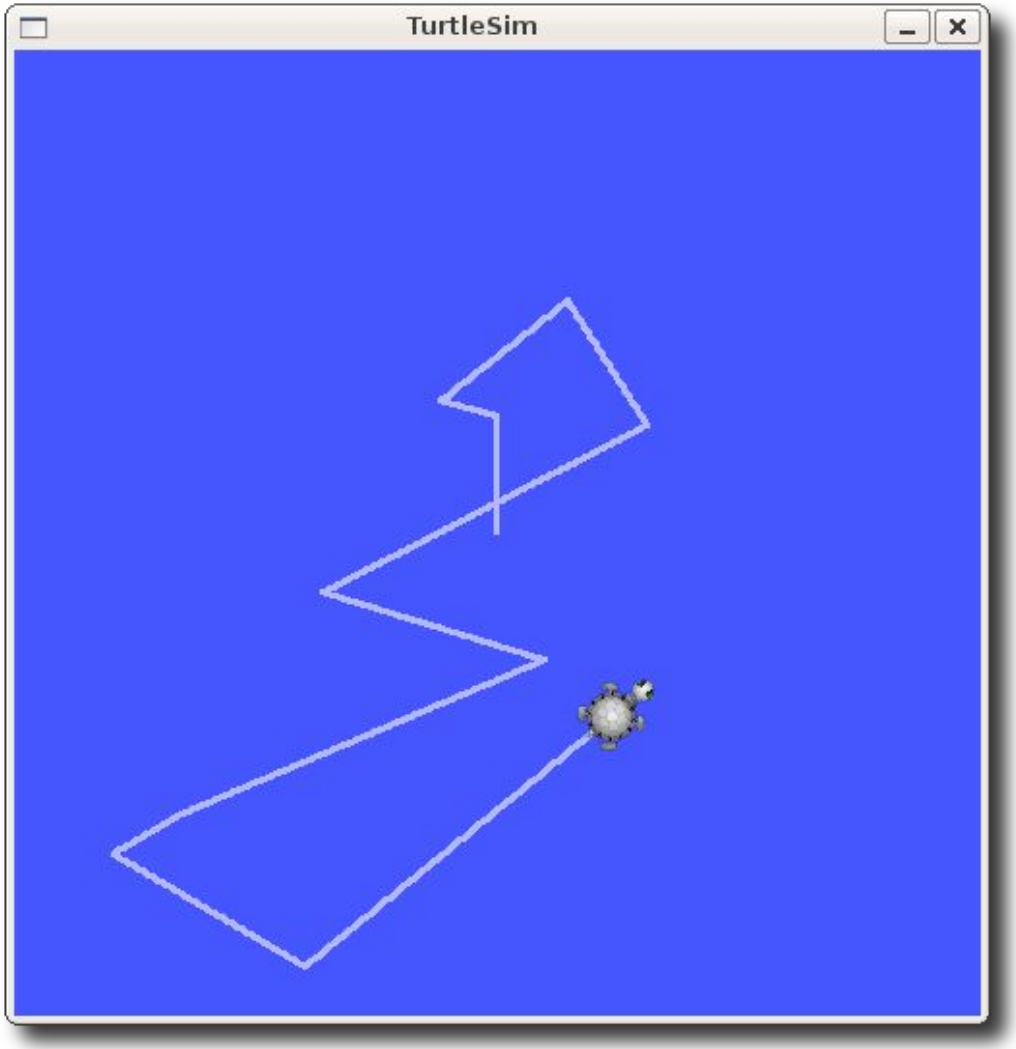
- roscore = ros+core : mestre + rosout (stdout/stderr) + servidor de parâmetros
- rosnode = ros+node : Ferramenta do ROS para pegar informações do nodo.
- rosrun = ros+run : roda um nó de um determinado pacote.

Turtle Sim

Vamos adicionar mais um nó agora!

Em um outro terminal coloque a seguinte linha de código

```
$ rosrun turtlesim turtle_teleop_key
```



- Tá… Beleza, mas o que que está acontecendo?
- Para ver isso aí precisamos de um outro recurso!

```
$ sudo apt-get install ros-<distro>-rqt

$ sudo apt-get install ros-<distro>-rqt-common-plugins
```

- Lembre-se de substituir distro pela versão _____ do ROS
- Em um outro terminal digite o comando:

```
$ rosrun rqt_graph rqt_graph
```

Vamos ver o que está sendo publicado no comado de velocidade do nó teleop_turtle

Em outro terminal digite:

```
$ rostopic echo /turtle1/cmd_vel
```

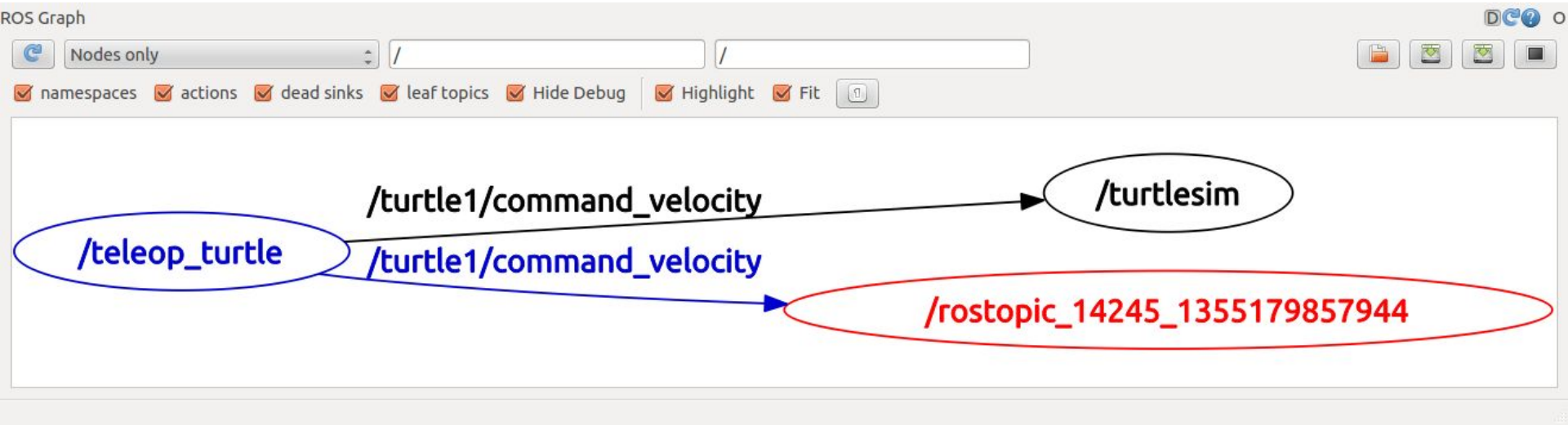
Você deve estar vendo isso aqui:

Agora selecione o terminal do teleop_turtle e aperte qualquer seta do teclado

Agora sim!
Deve estar aparecendo isso aqui:

```
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

Já deram uma olhada naquele gráfico de antes? Deem uma atualizada nele!



Olha outro nó aí!

Turtle Sim

Vamos dar uma olha no tipo dos tópicos agora, isto é, que tipo de mensagens eles compartilam, para isso vamos usar o comando:

```
$rostopic type [topic]
```

No nosso caso, vamos ver o que o cmd_vel publica, que tipo de mensagem ele troca:

```
$ rostopic type /turtle1/cmd_vel
```

Isso deve te retornar:

```
geometry_msgs/Twist
```

Turtle Sim

“geometry_msgs/Twist” …

Beleza… mas que que isso me diz?
É só perguntar pro ROS:

```
$ rosmsg show geometry_msgs/Twist
```

geometry_msgs/Vector3 linear

  float64 x

  float64 y

  float64 z

geometry_msgs/Vector3 angular

  float64 x

  float64 y

  float64 z

Opa! Agora eu sei o que ele pública!

Será que tem como mandar uma mensagem manualmente?

Turtle Sim

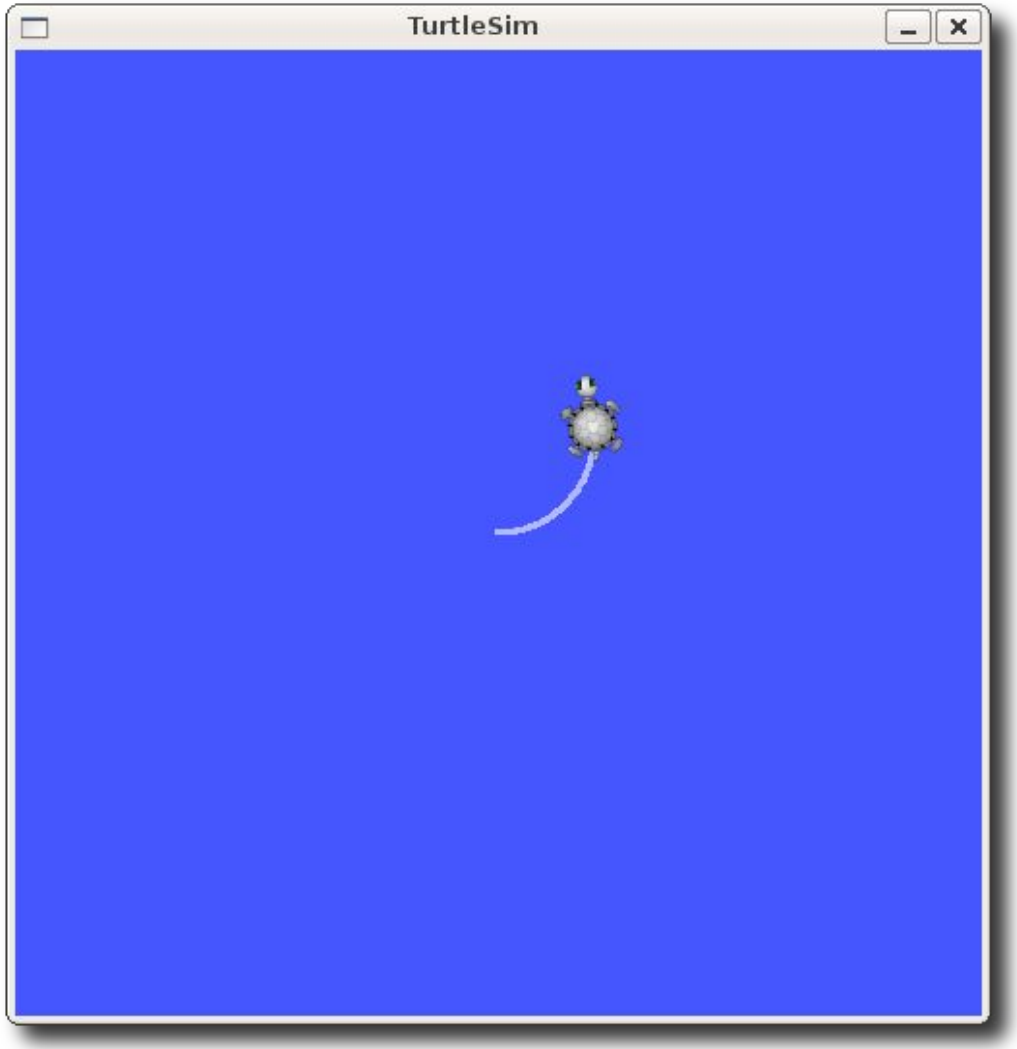
Não… Capaz, tem sim! O comando para isso é:

```
$rostopic pub [topico] [tipo_msg] [args]
```

Aqui no caso o comando fica:

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
```

Aí estamos mandando uma unica vez a tartaruguinha andar com uma velocidade linear de 2 e uma angula de 1.8



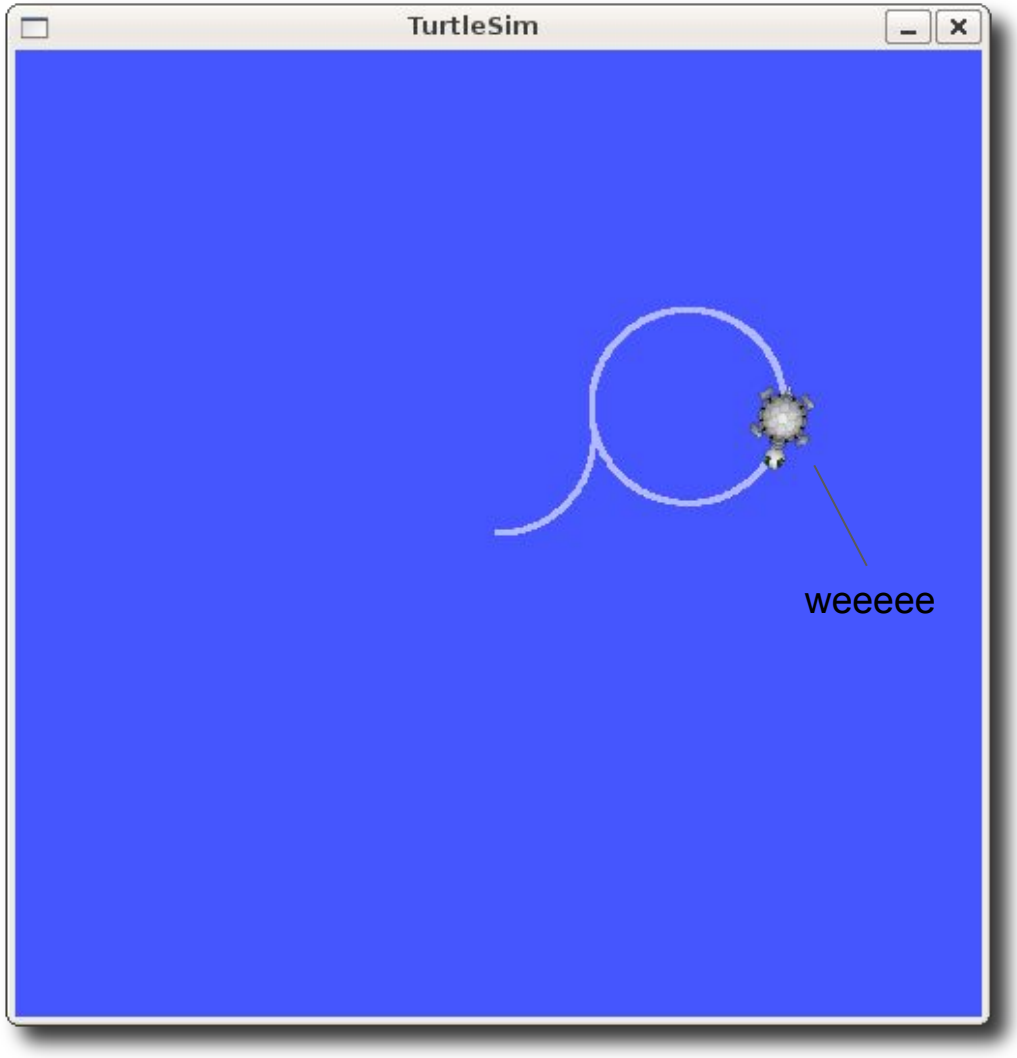
O que fizemos aqui?
rostopic pub - publica em um dado tópico

-1 -publica uma única vez

/turtle1/command_velocity - tópico em que vamos publicar

turtlesim/Velocity - tipo de mensagem que vamos publicar

A tartaruga parou não é?
Ela precisa de comandos contínuos sendo publicados a uma frequência de 1 Hz

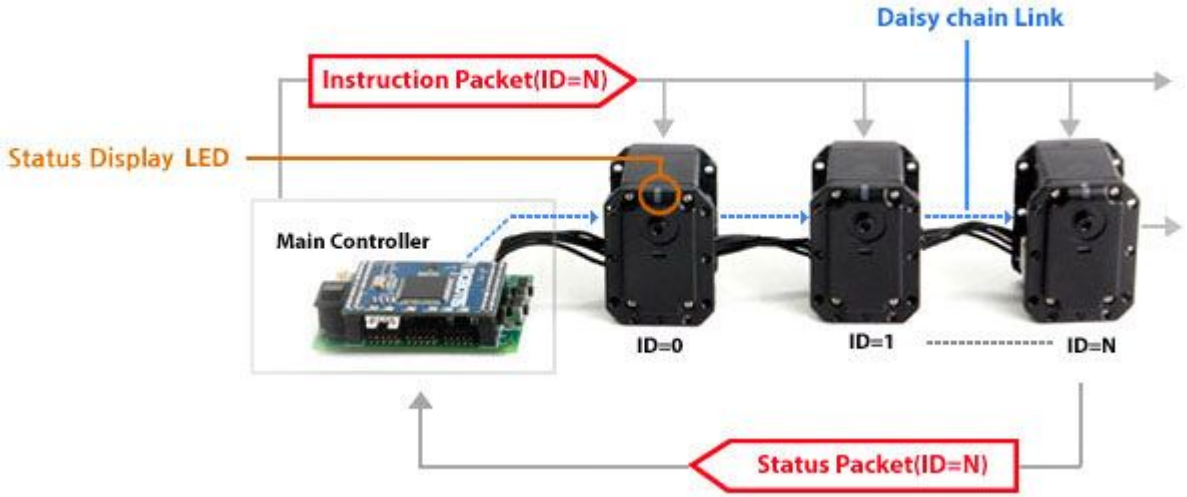

TurtleSim
weeeee

Turtle Sim

E é isso os nós que escutam e os nós que publicam, podemos publicar manualmente também. Como funcionam os tópicos do ROS e o porquê de eles terem tipos.

DEMO: Dynimexel & ROS

https://github.com/TauraBots/head_darwin_tb



ROS launch!

Com esta ferramenta podemos facilmente iniciar vários nós utilizando um arquivo de configuração XML.

```
<!-- -*- mode: XML -*- -->
<launch>
    <node name="dynamixel_manager" pkg="dynamixel_controllers" type="controller_manager.py" required="true" output="screen">
        <rosparam>
            namespace: dxl_manager
            serial_ports:
                pan_tilt_port:
                    port_name: "/dev/ttyUSB0"
                    baud_rate: 1000000
                    min_motor_id: 1
                    max_motor_id: 25
                    update_rate: 20
        </rosparam>
    </node>
</launch>
```

ROS launch!

```
<!-- -*- mode: XML -*- -->
<launch>
    <!-- Start tilt and pan controller -->
    <rosparam file="$(find head_darwin_tb)/param/parm_head.yaml" command="load"/>
    <node name="tilt_controller_spawner" pkg="dynamixel_controllers" type="controller_spawner.py"
          args="--manager=dxl_manager
                --port pan_tilt_port
                head_tilt_controller
                head_pan_controller"
          output="screen"/>

  <!-- Start joints trajectory controller controller -->
    <rosparam file="$(find head_darwin_tb)/param/parm_head.yaml" command="load"/>
    <node name="controller_spawner_meta" pkg="dynamixel_controllers" type="controller_spawner.py"
          args="--manager=dxl_manager
                --type=meta
                f_head_controller
                head_pan_controller
                head_tilt_controller"

                output="screen"/>
</launch>
```

Let’s Move it!

```
#Primeiro inicializamos o controlador dos motores
roslaunch head_darwin_tb dx_controller.launch
#Inicializamos os parâmetros da cabeça do darwin
roslaunch head_darwin_tb start_controller.launch
```

Agora podemos publicar nos tópicos criados

```
#Move a cabeça aproximadamente 28 graus
rostopic pub -1 /head_pan_controller/command std_msgs/Float64 -- -0.5
#Desabilita o torqe nos motores
rosservice call /head_pan_controller/torque_enable False
```