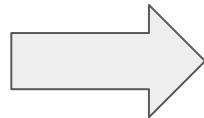




# ROBOT OPERATING SYSTEM (ROS)

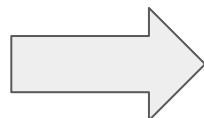
Kristofer Stift Kappel  
[kskappel@inf.ufpel.edu.br](mailto:kskappel@inf.ufpel.edu.br)

# Repositório



<https://github.com/kriskappel/MiniCursoRos>

## Tutorial baseado nos links:



<http://wiki.ros.org/ROS/Introduction>

<http://wiki.ros.org/ROS/Tutorials>

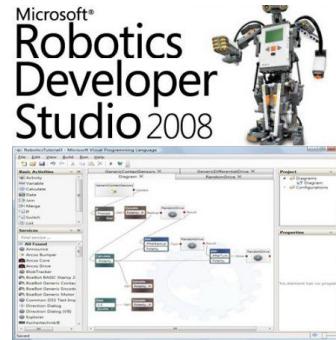
# Introdução



ROS

THOR

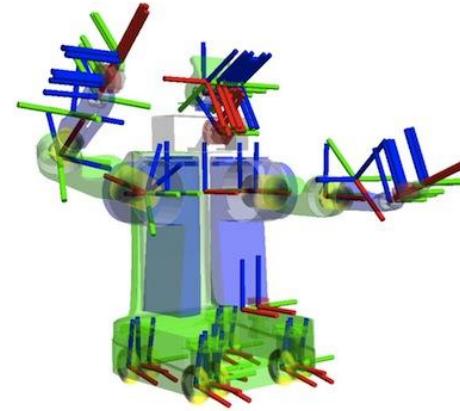
Carmen  
Robot Navigation Toolkit



# Por que usar o ROS?

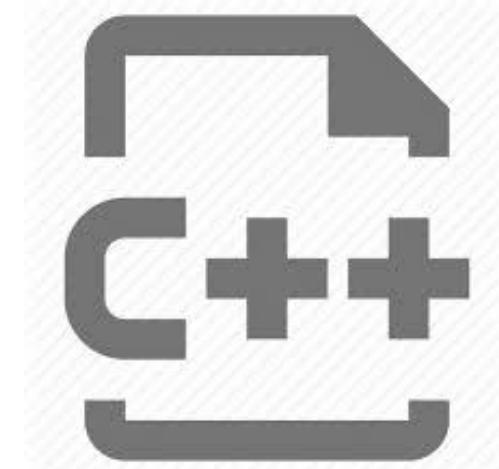
ROS

- ROS Topic
- ROS Bag
- ROS Service
- ROS Param
- Padronização
- TF
- URDF
- Github



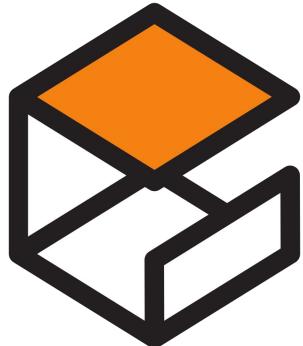
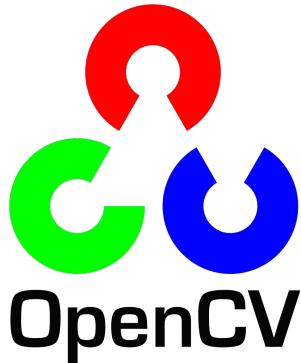
# Por que usar o ROS?

- Reutilização de código

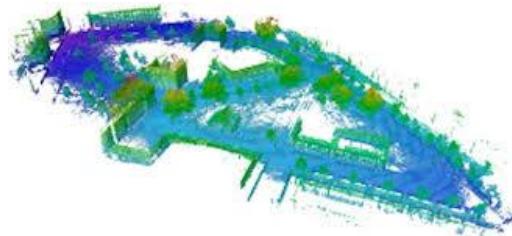
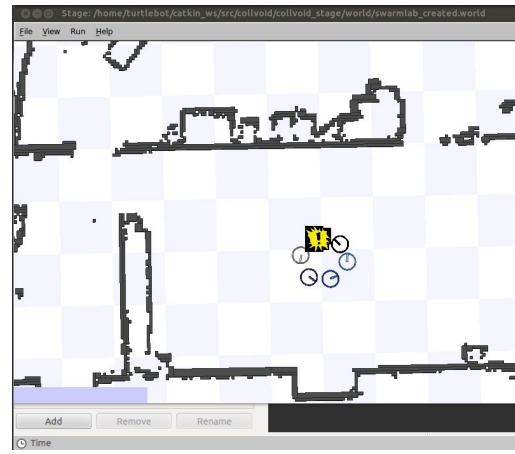


# Por que usar o ROS?

- Várias bibliotecas e simuladores



Eigen, KDL, ...



# Por que usar o ROS?

- Robôs:



<https://robots.ros.org/all/>

# O que é o ROS?

- Meta-Operating System!



# Distribuições do ROS



Box Turtle

Box Turtle: 2010 - Ubuntu 8.04 (Hardy)



C Turtle: 2 de Agosto de 2010 - Ubuntu 9.04 (Jaunty)



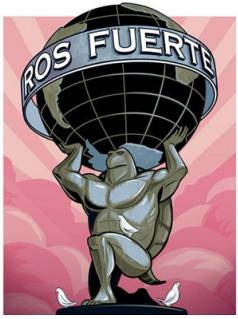
DiamondBack: 2011 - Ubuntu 10.04 (Lucid)



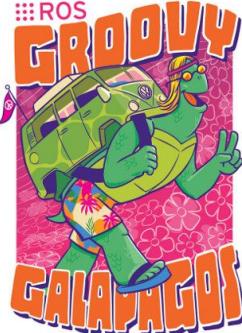
Electric Emys: 2011 - Ubuntu 10.04 (Lucid)



# Distribuições do ROS



Fuerte Turtle: 2012 - Ubuntu 10.04, 11.10 e 12.04 (Lucid, Oneric e Precise)



Groovy Galapagos: 2012 - Ubuntu 12.04 (Precise)



Hydro Medusa: 2013 - Ubuntu 12.04 (Precise)

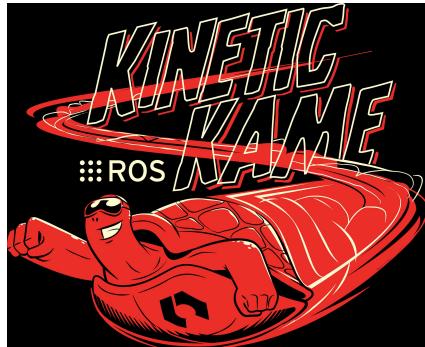


Indigo Igloo: 2014 - Ubuntu 14.04 (Trusty)

# Distribuições do ROS



Jade Turtle: 2015 - Ubuntu 15.04 (Vivid)



Kinetic Kame: 2016 - Ubuntu 16.04 (Xenial)

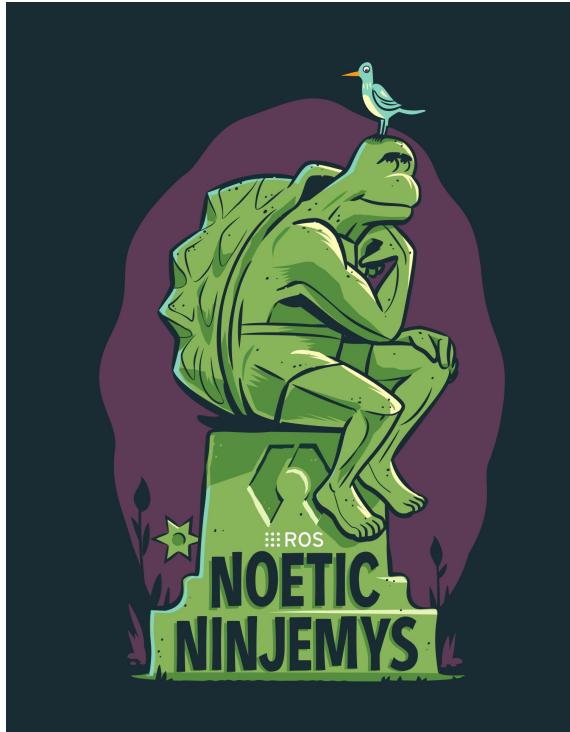


Lunar Loggerhead: 2017 - Ubuntu 17.04 (Zesty)



Melodic Morenia: 2018 - Ubuntu 18.04 (Bionic)

# Distribuições do ROS



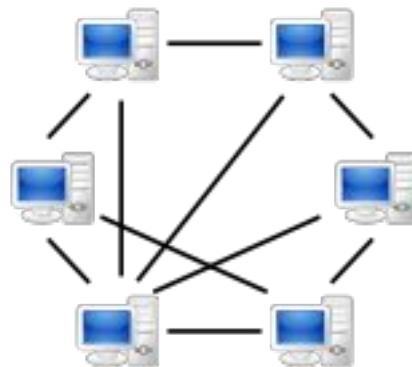
Noetic Ninjemys: 2020 - Ubuntu 20.04 (Focal)

# Workspace

```
workspace_folder/          -- WORKSPACE
  src/                      -- SOURCE SPACE
    CMakeLists.txt           -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      CMakeLists.txt
      package.xml
      ...
  build/                   -- BUILD SPACE
    CATKIN_IGNORE            -- Keeps catkin from walking this directory
  devel/                   -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
  install/                 -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
```

# Conceitos do ROS

- Grafo de computação:



p2p

# Conceitos do ROS

Master

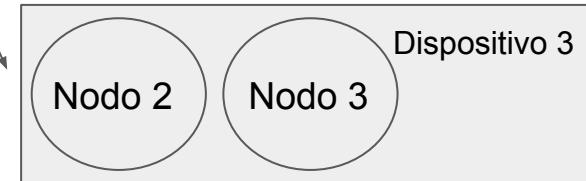
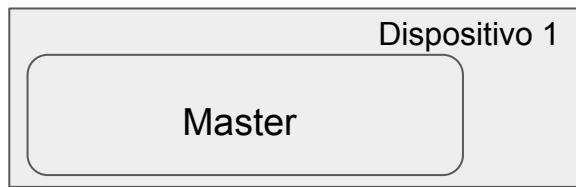
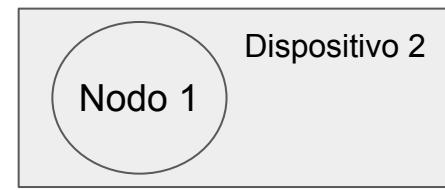
Node

Parâmetros

Tópicos

Serviços

Bags



# Conceitos do ROS

Master

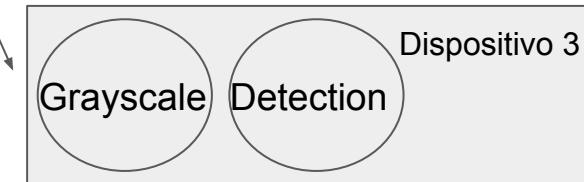
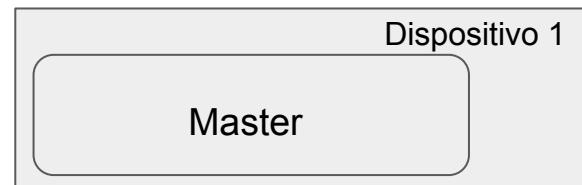
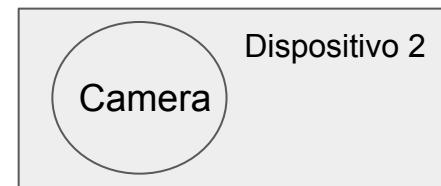
**Node**

Parâmetros

Tópicos

Serviços

Bags



# Conceitos do ROS

Master

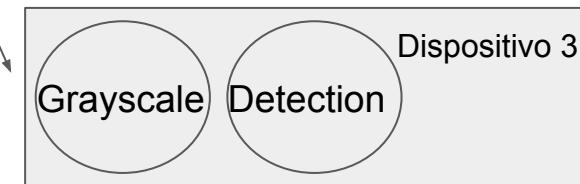
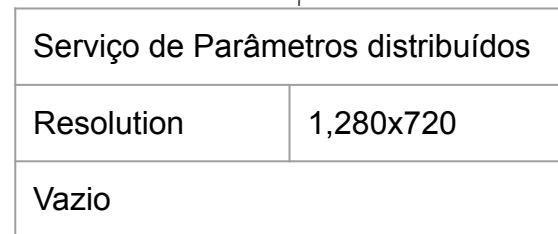
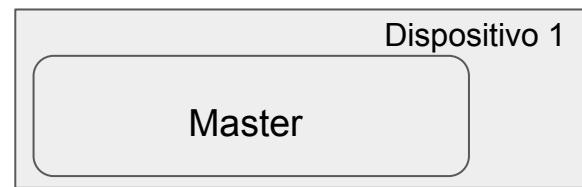
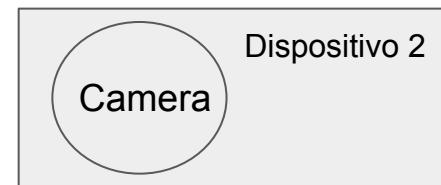
Node

**Parâmetros**

Tópicos

Serviços

Bags



# Conceitos do ROS

Master

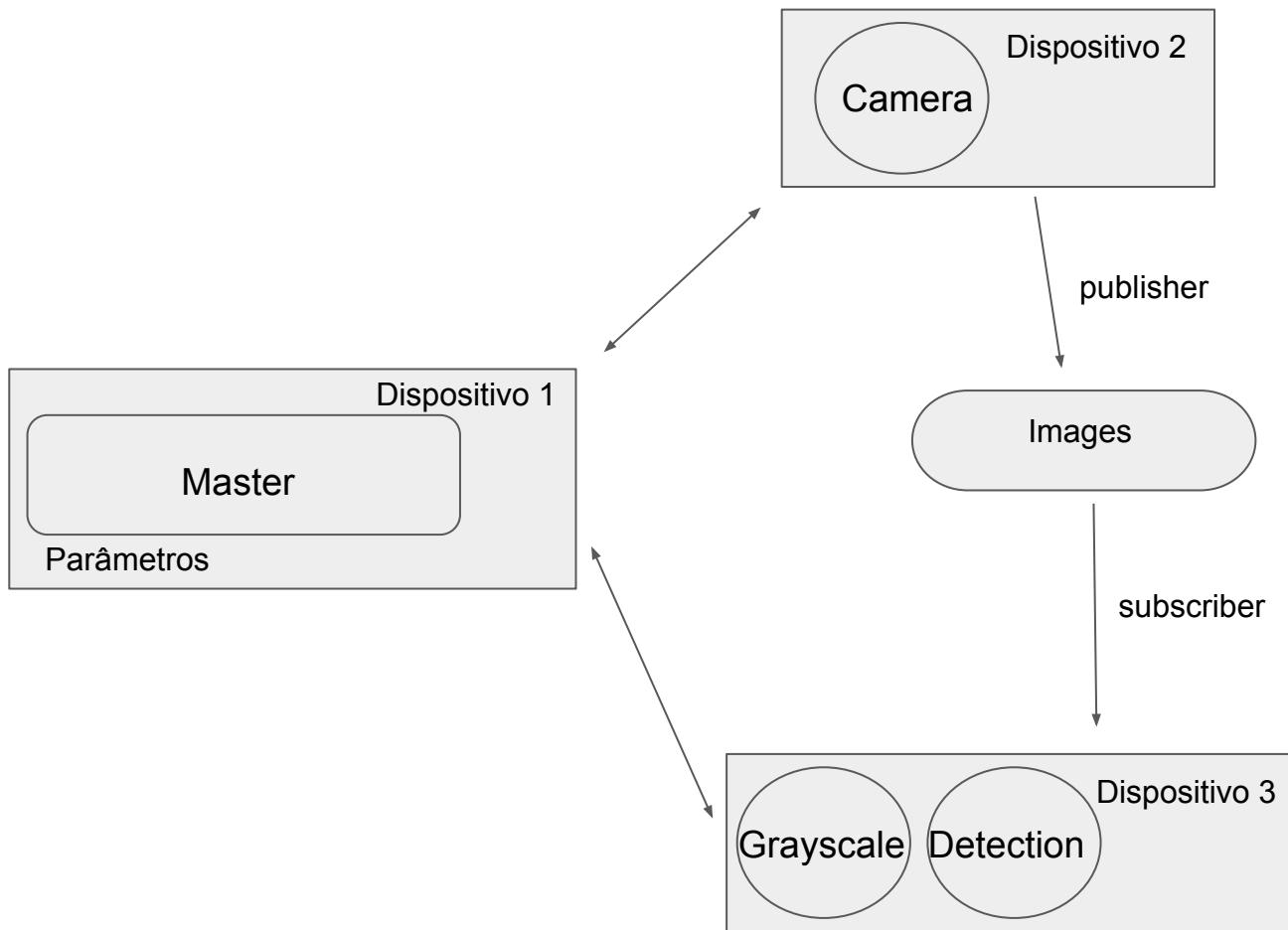
Node

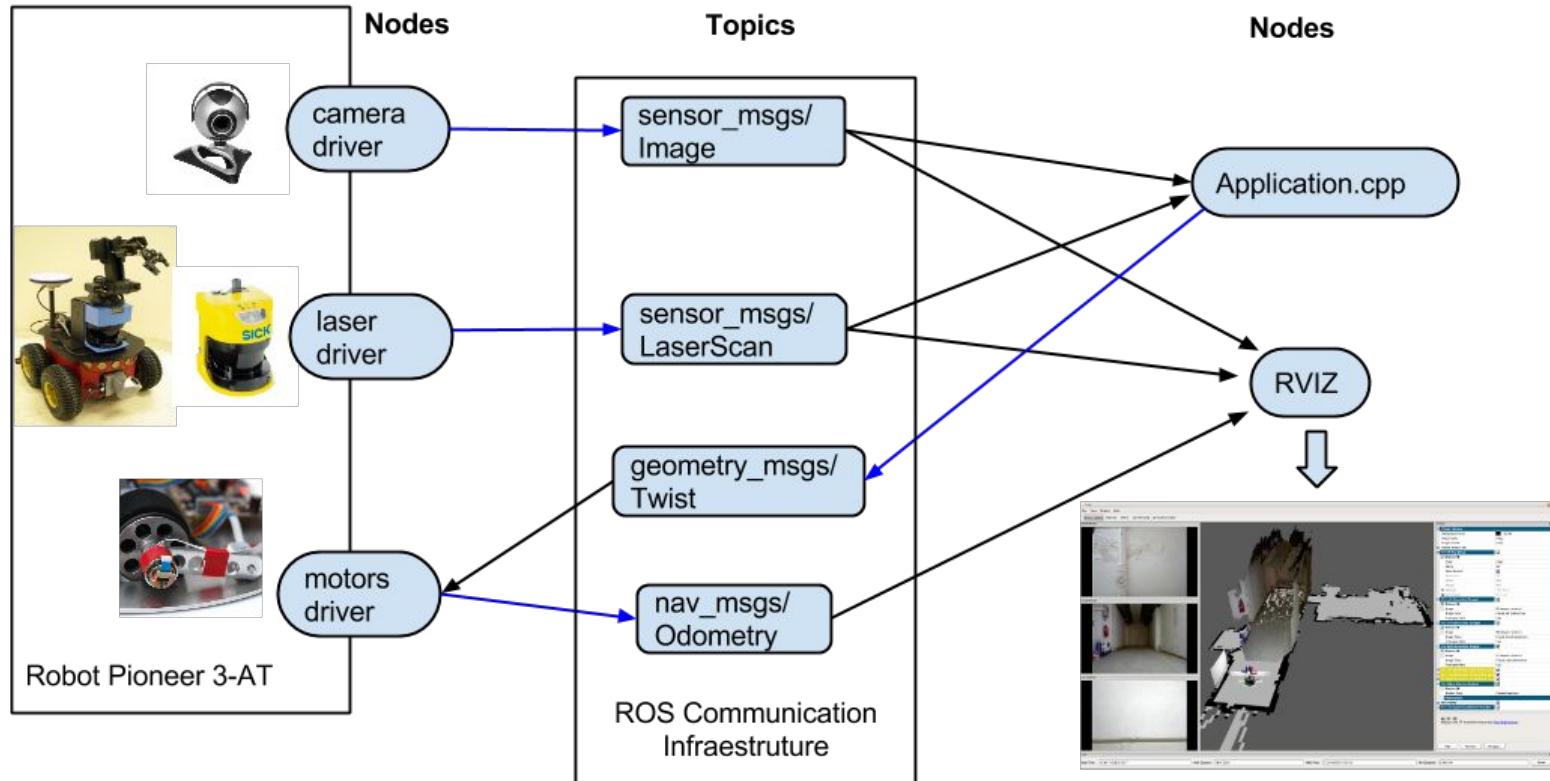
Parâmetros

**Tópicos**

Serviços

Bas





# Conceitos do ROS

Master

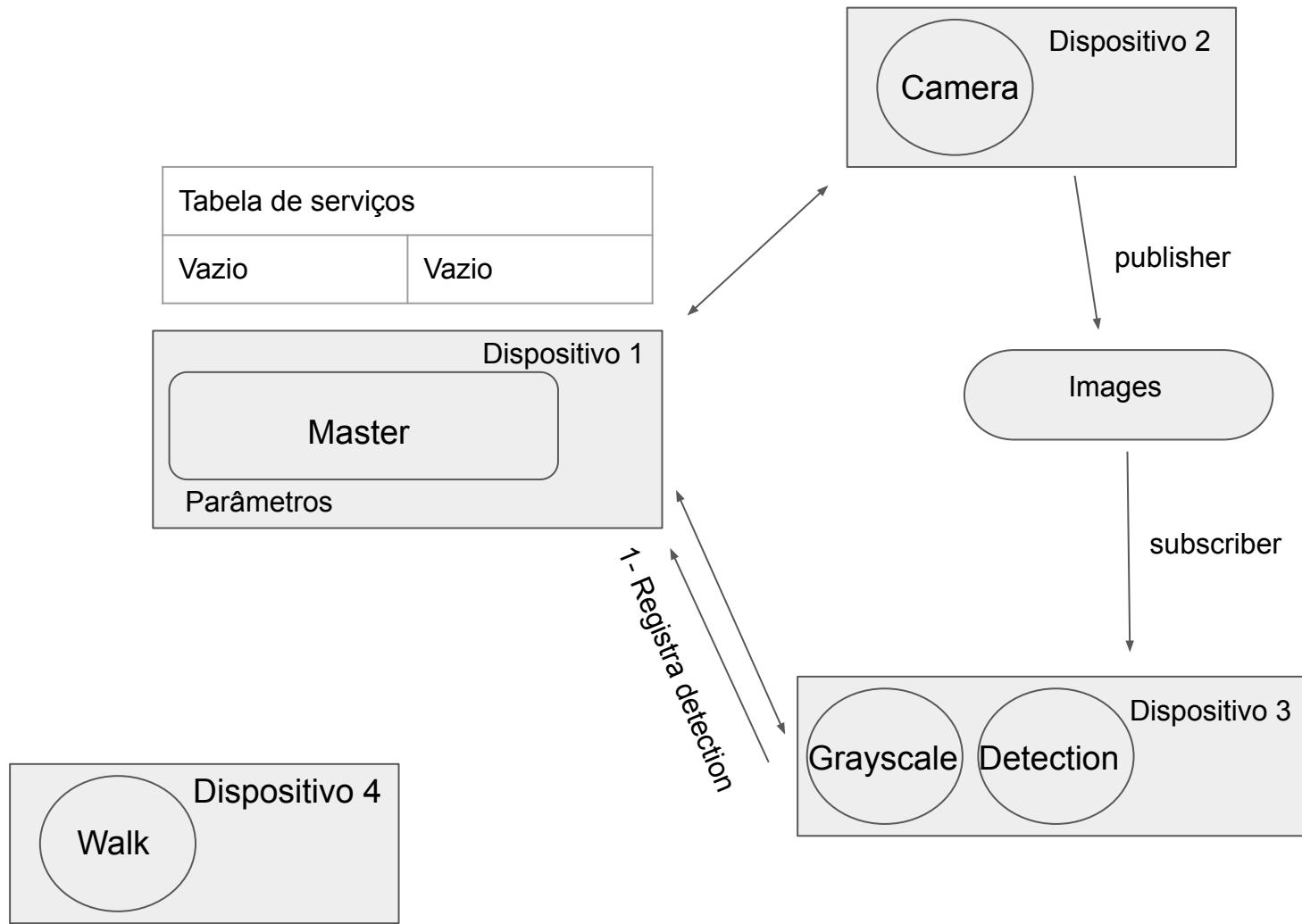
Node

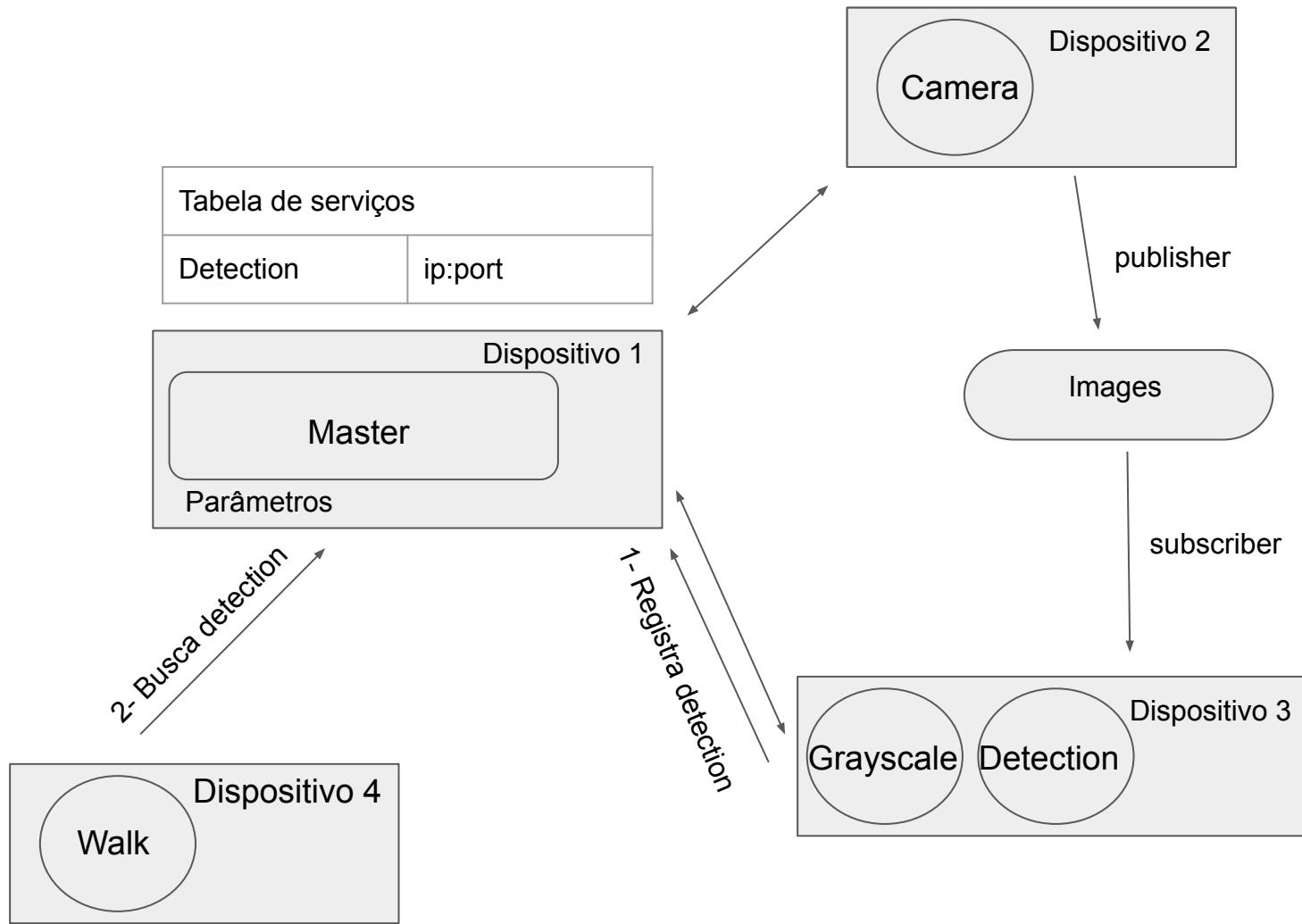
Parâmetros

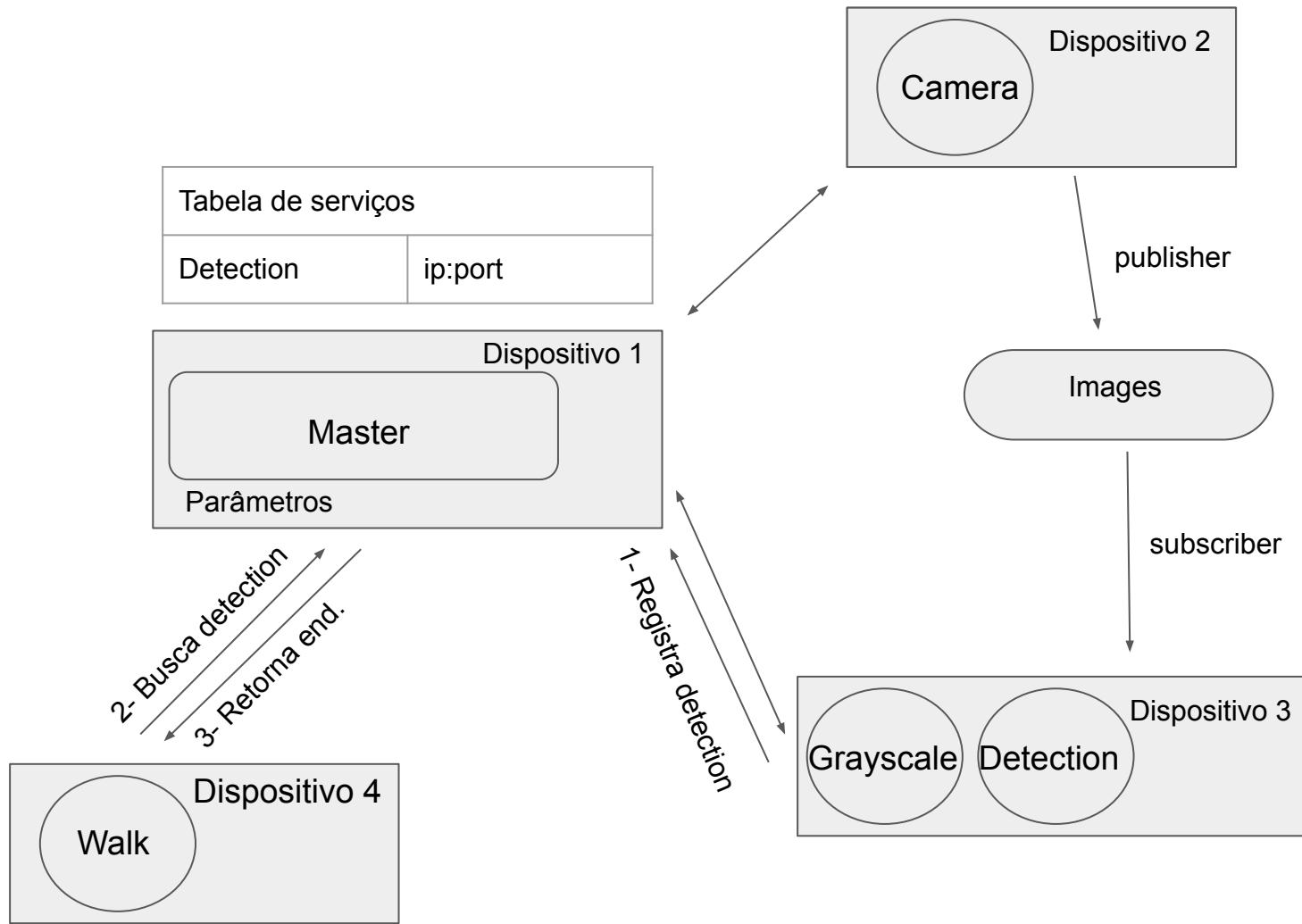
Tópicos

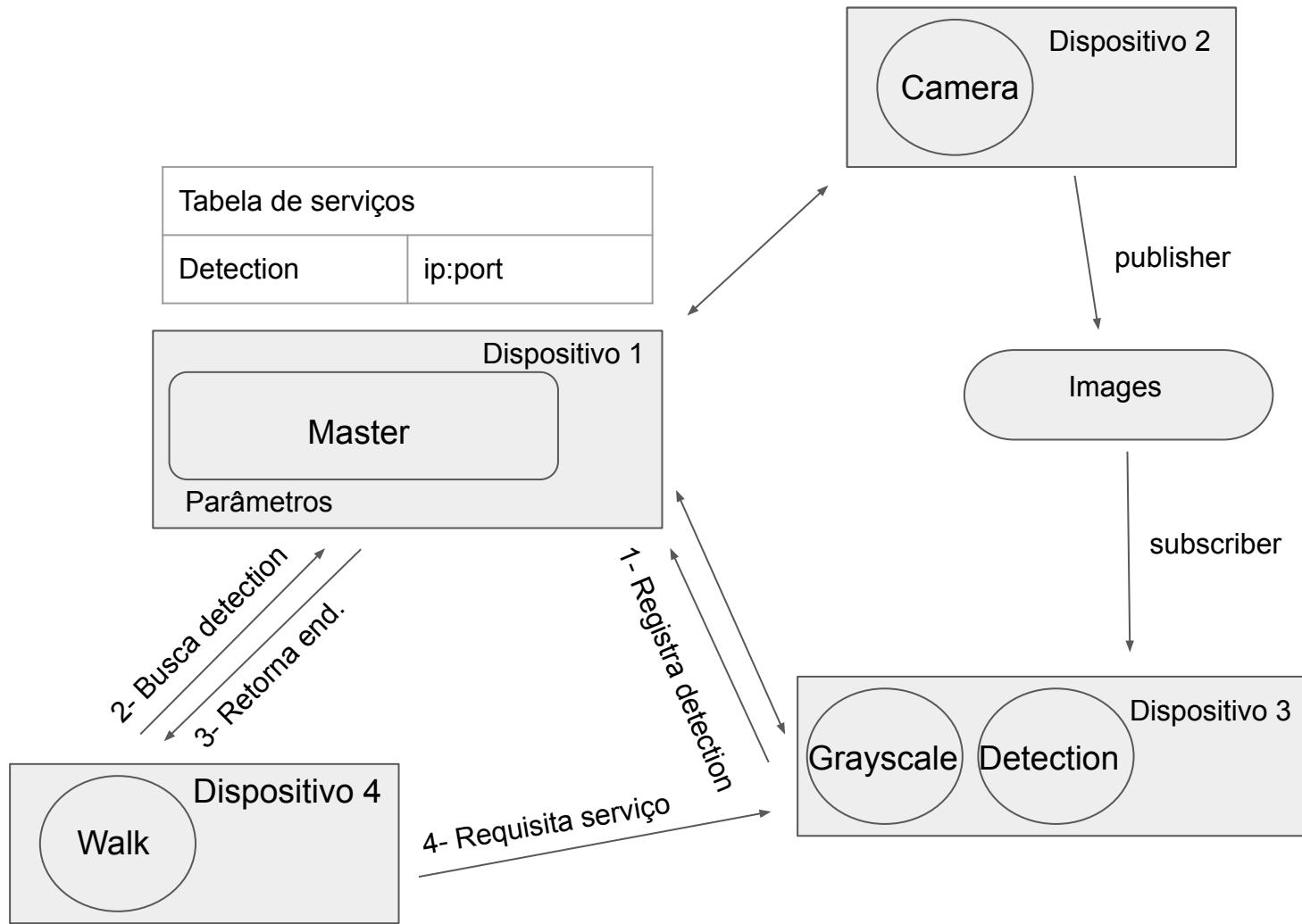
**Serviços**

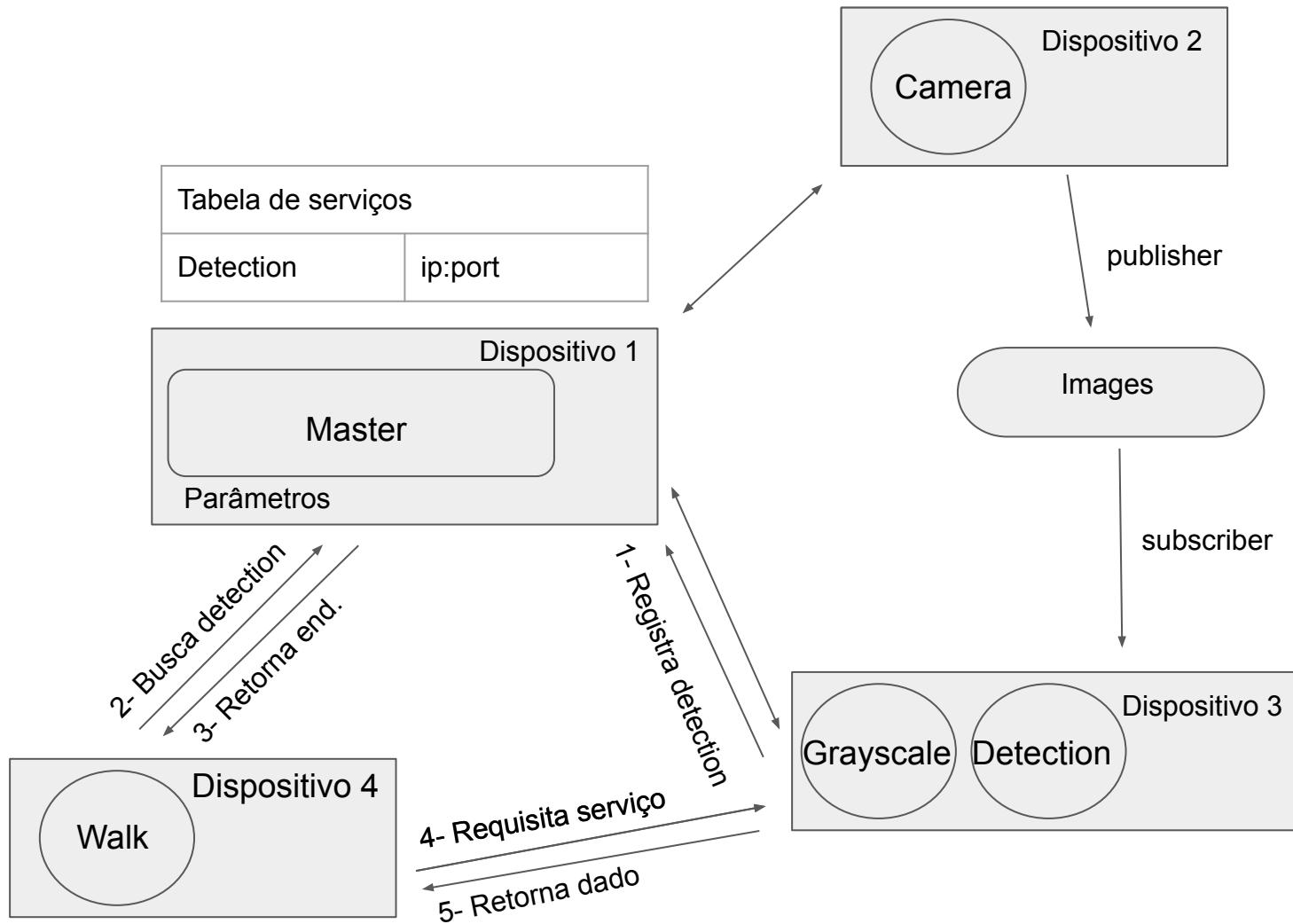
Bags











# Conceitos do ROS

Master

Node

Parâmetros

Tópicos

Serviços

Bags



# Instalando o ROS

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

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key  
421C365BD9FF1F717815A3895523BAEEB01FA116
```

```
sudo apt-get update
```

```
sudo apt-get install ros-kinetic-desktop-full
```

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

```
source ~/.bashrc
```

# Mãos à obra

Inicie o Master!

```
$ roscore
```

Abra um novo terminal e verifique os nodos que estão em execução:

```
$ rosnode list
```

# Mãos à obra

- Brincando com o turtlesim:

Execute o turtlesim:

```
$ rosrun turtlesim turtlesim_node
```

Abra um novo terminal e verifique os nodos que estão em execução:

```
$ rosnodes list
```



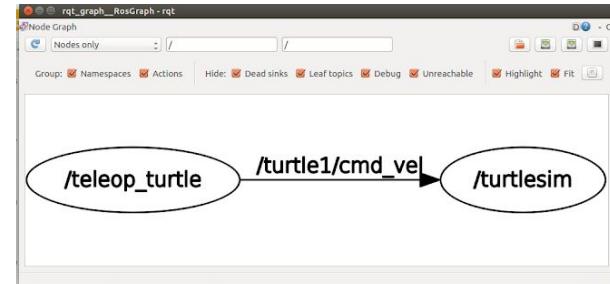
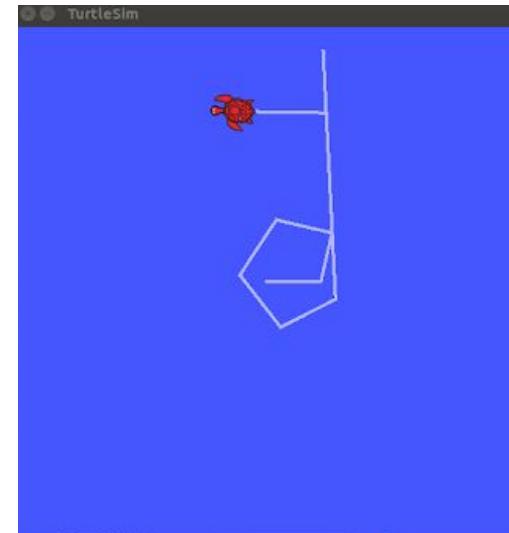
# Mãos à obra

Agora controle a tartaruga! Em um novo terminal  
digite:

```
$ rosrun turtlesim turtle_teleop_key
```

É possível também ver a arquitetura do ROS  
digitando:

```
$ rosrun rqt_graph rqt_graph
```



# Mãos à obra

Para ver o que está sendo publicado no tópico /turtle1/cmd\_vel, digitem:

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

Para ver o tipo da mensagem que está sendo publicada é necessário digitar:

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

Para descobrir os tipos que constituem as mensagens digitem:

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

# Mãos à obra

Para ver o que está sendo publicado no tópico /turtle1/cmd\_vel, digitem:

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

Para ver o tipo da mensagem que está sendo publicada é necessário digitar:

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

Para descobrir os tipos que constituem as mensagens digitem:

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

# Mãos à obra

- Publicando mensagens:

```
rostopic pub [tópico] [tipo da mensagem] [mensagem]
```

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

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

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

# Mãos à obra

Para medir a frequência de publicação das mensagens basta digitar:

```
$ rostopic hz /turtle1/pose
```

Para plotar os dados de um tópico:

```
$ rosrun rqt_plot rqt_plot
```

# Mãos à obra

- Trabalhando com serviços!

Para listar os serviços disponíveis:

```
$ rosservice list
```

Para descobrir o tipo de um serviço:

```
$ rosservice type /clear
```

Para chamar um serviço basta digitar rosservice call [serviço][argumentos]

```
$ rosservice call /clear
```

# Mãos à obra

- Chamando as tartarugas ninjas!

Descubra a definição do serviço:

```
$ rosservice type /spawn | rossrv show
```

Mate a tartaruga existente :(

```
$ rosservice call /kill turtle1
```

# Mãos à obra

Agora adicione as novas tartarugas!

```
rosservice call /spawn 4 5 0.0 'Donatelo'
```

```
rosservice call /spawn 6 5 3.0 'Michelangelo'
```

```
rosservice call /spawn 5 4 1.5 'Leonardo'
```

```
rosservice call /spawn 5 6 4.7 'Rafael'
```

Para controlar a nova tartaruga:

```
rosrun turtlesim turtle_teleop_key /turtle1/cmd_vel:=*Nome da turtle*/cmd_vel
```

# Mãos à obra

- Trabalhando com parâmetros

Para listar os parâmetros:

```
$ rosparam list
```

Consultando parâmetro:

```
$ rosparam get [nome do parâmetro] (se deixar só um "/" consulta todos ao mesmo tempo)
```

Alterando parâmetros

```
$ rosparam set [nome do parâmetro]
```

# Mãos à obra

Consulte o componente verde da cor de fundo do turtlesim:

```
$ rosparam get /background_g
```

Altere a cor de fundo do simulador

```
$ rosparam set /background_r 150
```

Para atualizar digite:

```
$ rosservice call /clear
```

# Mãos à obra

- Revisão:

roscore - Inicia o master;

rosrun - Executa um nodo;

rosnodes - Funções relacionadas a nodos;

rosservice - Funções relacionadas a serviços;

rostopic - Funções relacionadas a tópicos;

rosparam - Funções relacionadas a parâmetros;

# Mãos à obra

- Workspace:

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

Criando o workspace:

```
$ catkin_init_workspace
```

Compilando o workspace:

```
$ cd ..
```

```
$ catkin_make
```

# Mãos à obra

- Pacotes

Acesse a pasta:

```
$ cd ~/catkin_ws/src
```

Execute o script para a criação:

```
#catkin_create_pkg <Nome do Pacote> [Dependências]
```

```
$ catkin_create_pkg minicurso std_msgs rospy roscpp
```

Para compilar:

```
cd ~/catkin_ws && catkin_make
```

# Mãos à obra

Último passo é adicionar a pasta catkin\_ws à seu bashrc

```
$ cd ~/catkin_ws
```

```
$ echo 'source ~/catkin_ws/devel/setup.bash' >> ~/.bashrc
```

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

# Mãos à obra

- Launch

Próximo passo é a construção de um arquivo launch para automatizar a execução de vários nodos.

Entre na pasta do pacote:

```
$ cd ~/catkin_ws/src/minicurso/src
```

Crie uma pasta para guardar seus arquivos launch:

```
$ mkdir launch && cd launch
```

Na pasta crie seus arquivos launch (por exemplo “teste.launch”) com o conteúdo do slide seguinte.

# Mãos à obra

- Launch Exemplo 1

Para executar digite roslaunch [nome do pacote][nome do arquivo.launch]

```
<launch>
  <node pkg="turtlesim" name="simulador" type="turtlesim_node"/> <!--abre turtlesim-->

  <node pkg="rostopic" name="publisher" type="rostopic"
        args="pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]"/>
  <!--publica no tópico-->

  <node pkg="rosservice" name="tinta" type="rosservice" args="call /turtle1/set_pen 112 112 112 10
0"/> <!--troca a cor da tinta-->

</launch>
```

# Mãos à obra

- Launch Exemplo 2: Abrindo 2 turtlesim e controlando ao mesmo tempo.

```
<launch>
    <group ns="turtlesim1">
        <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
    </group>
    <group ns="turtlesim2">
        <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
    </group>

    <node pkg="turtlesim" name="mimic" type="mimic">
        <remap from="input" to="turtlesim1/turtle1"/>
        <remap from="output" to="turtlesim2/turtle1"/>
    </node>
</launch>
```

Para controlar a turtle: rosrun turtlesim turtle\_teleop\_key turtle1/cmd\_vel:=/turtlesim1/turtle1/cmd\_vel

# Mãos à obra

- Launch Exemplo 3: Os launches não garantem a execução em ordem. Serviços que NECESSITAM previamente da execução de outros nodos podem ser separados em mais de um launch.

Arquivo 1 - Abre o turtlesim.

```
<launch>
    <node pkg="turtlesim" name="simulador" type="turtlesim_node"/> <!--abre turtlesim-->
</launch>
```

Arquivo 2- Depois do turtlesim ja aberto ele mata a turtle existente e cria novas.

```
<launch>
    <node pkg="rosservice" name="mataTurtle" type="rosservice" args="call /kill turtle1"/> <!--mata a
turtle existente-->

    <!--cria as tartarugas ninja-->
    <node pkg="rosservice" name="mike" type="rosservice" args="call /spawn 4 5 0.0 Michelangelo "/>
    <node pkg="rosservice" name="rapha" type="rosservice" args="call /spawn 5 6 4.7 Rafael "/>
    <node pkg="rosservice" name="donnie" type="rosservice" args="call /spawn 6 5 3.2 Donatelo "/>
    <node pkg="rosservice" name="leo" type="rosservice" args="call /spawn 5 4 1.5 Leonardo "/>
</launch>
```

# Código Publisher/Subscriber C++

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <iostream>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "talker");
    ros::NodeHandle n;
    ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
    ros::Rate loop_rate(10);
    int count = 0;
    while (ros::ok())
    {
        std_msgs::String msg;
        std::stringstream ss;
        ss << "hello world " << count;
        msg.data = ss.str();
        ROS_INFO("%s", msg.data.c_str());
        chatter_pub.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
        ++count;
    }
    return 0;
}
```

# Código Publisher/Subscriber C++

```
#include "ros/ros.h"
#include "std_msgs/String.h"
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}
int main(int argc, char **argv)
{
    ros::init(argc, argv, "listener");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
    ros::spin();
    return 0;
}
```

# Mãos à obra

Para pode chamar o talker e o listener de qualquer lugar é só adicionar no CMakeLists do pacote.

Adicionar os nodos no cmake:

- add\_executable(talker\_node src/talker.cpp)
- add\_executable(listener\_node src/listener.cpp)

Adicionar também as dependências padrões:

- target\_link\_libraries(talker\_node \${catkin\_LIBRARIES})
- target\_link\_libraries(listener\_node \${catkin\_LIBRARIES})

Compilar novamente: \$ cd ~/catkin\_ws , \$ catkin\_make

# Mãos à obra

Para executar os nodos no pacote.

Terminal 1:

```
$ roscore
```

Terminal 2:

```
$ rosrun minicurso talker_node
```

Terminal 3:

```
$ rosrun minicurso listener_node
```



# ROBOT OPERATING SYSTEM (ROS)

Kristofer Stift Kappel  
[kskappel@inf.ufpel.edu.br](mailto:kskappel@inf.ufpel.edu.br)