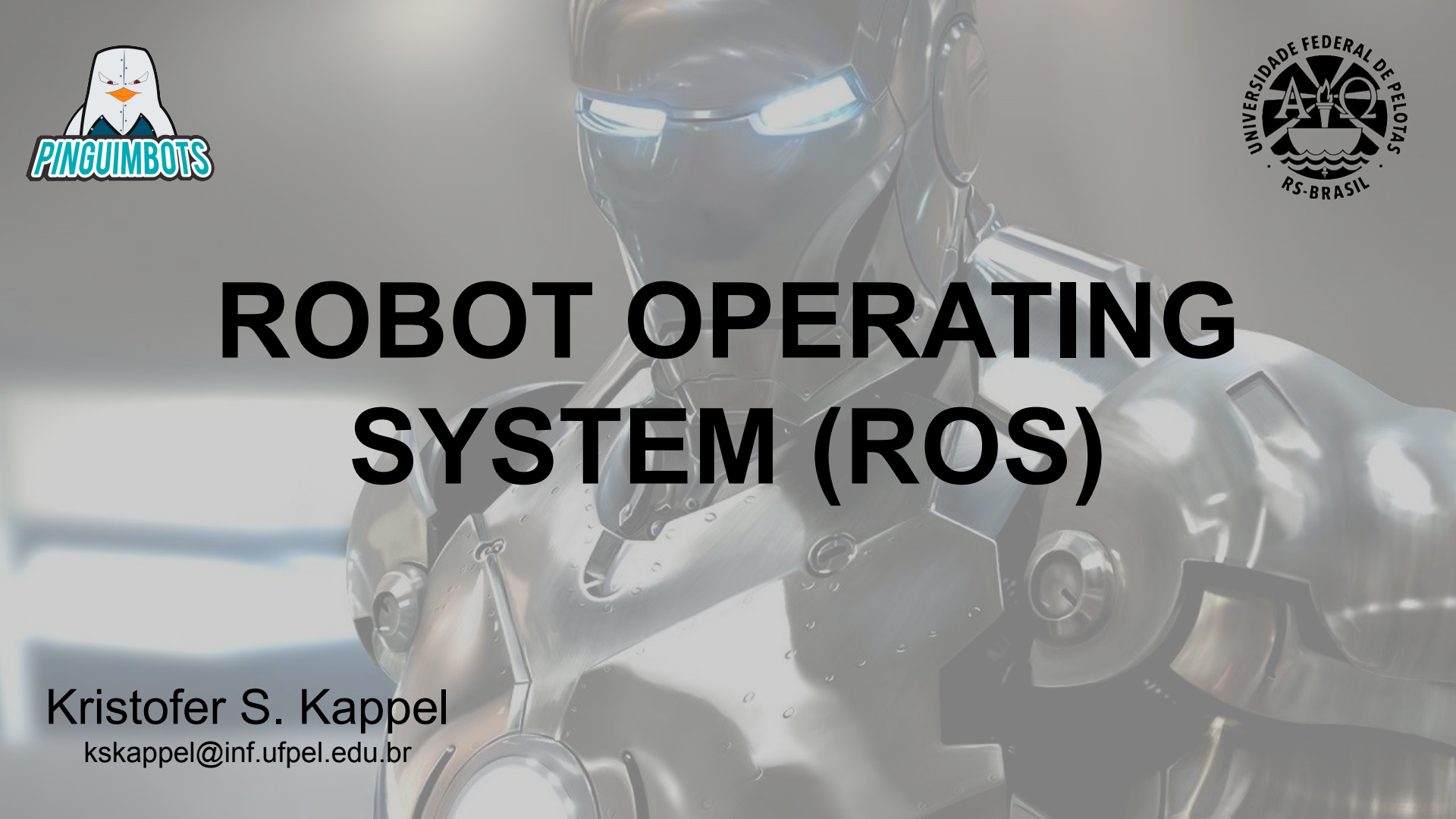
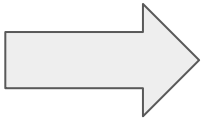


ROBOT OPERATING SYSTEM (ROS)

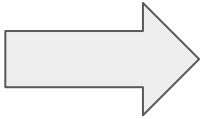
Kristofer S. Kappel
kskappel@inf.ufpel.edu.br

Repositório



https://github.com/kriskappel/MiniCursoRos

Tutorial baseado nos links:



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

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

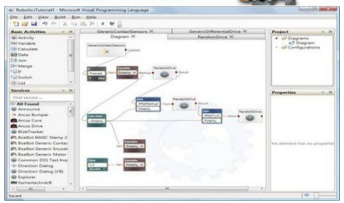
Introdução

ROS

YARP

Carmen
Robot Navigation Toolkit

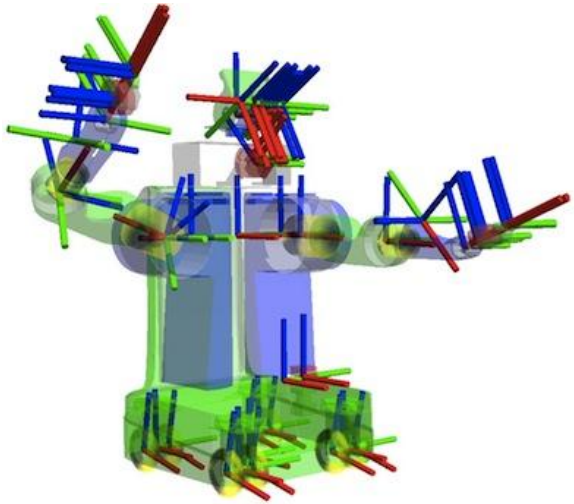
Microsoft®
Robotics
Developer
Studio 2008



Quais as vantagens do ROS?

ROS

- Padronização
- TF
- URDF
- Github

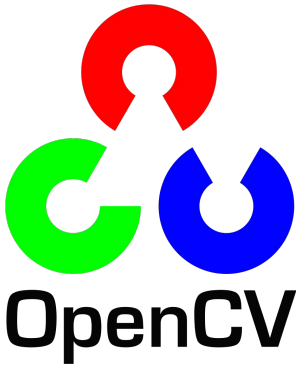


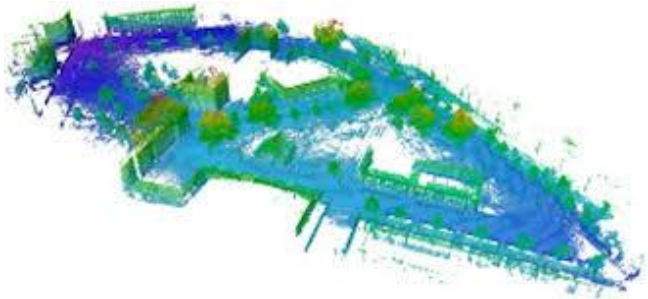
Quais as vantagens do ROS?

- Reutilização de código

Quais as vantagens do ROS?

- Várias bibliotecas e simuladores

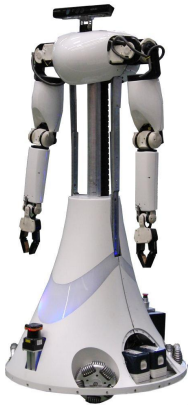




Quais as vantagens do ROS?

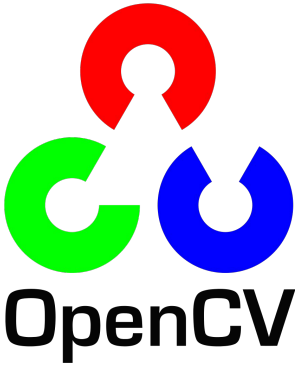
https://robots.ros.org/all/

- Robôs:

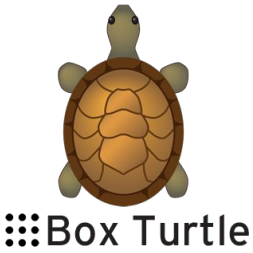


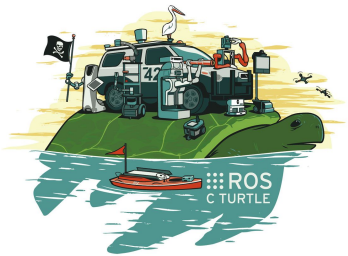
O que é o ROS?

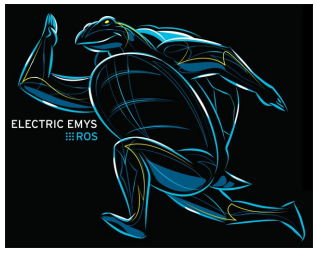
- Meta-Operating System!

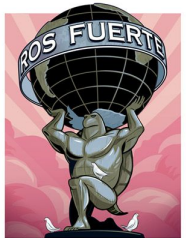


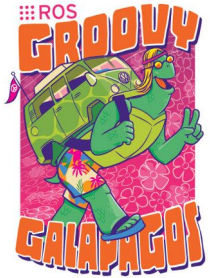
História, Distribuições do ROS e ROS 2



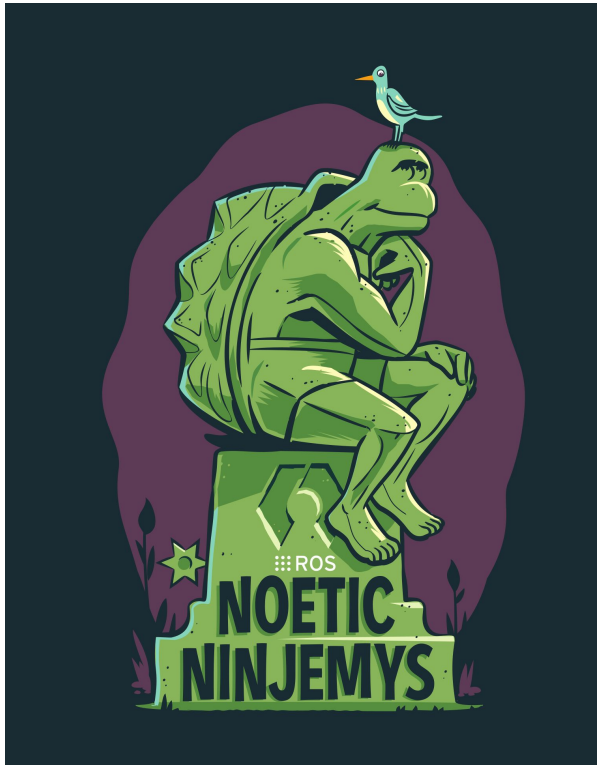


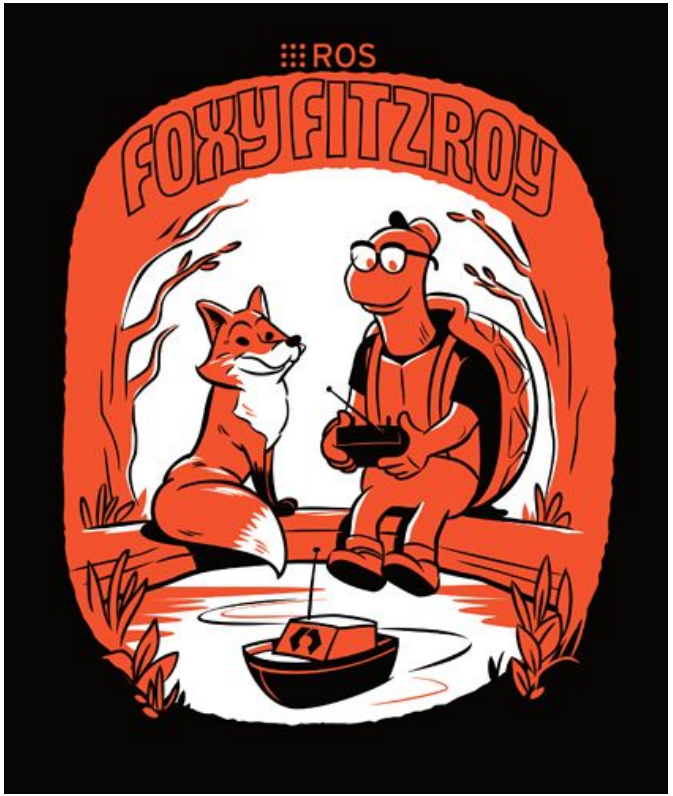






História, Distribuições do ROS e ROS 2





- **Master**

Nodo

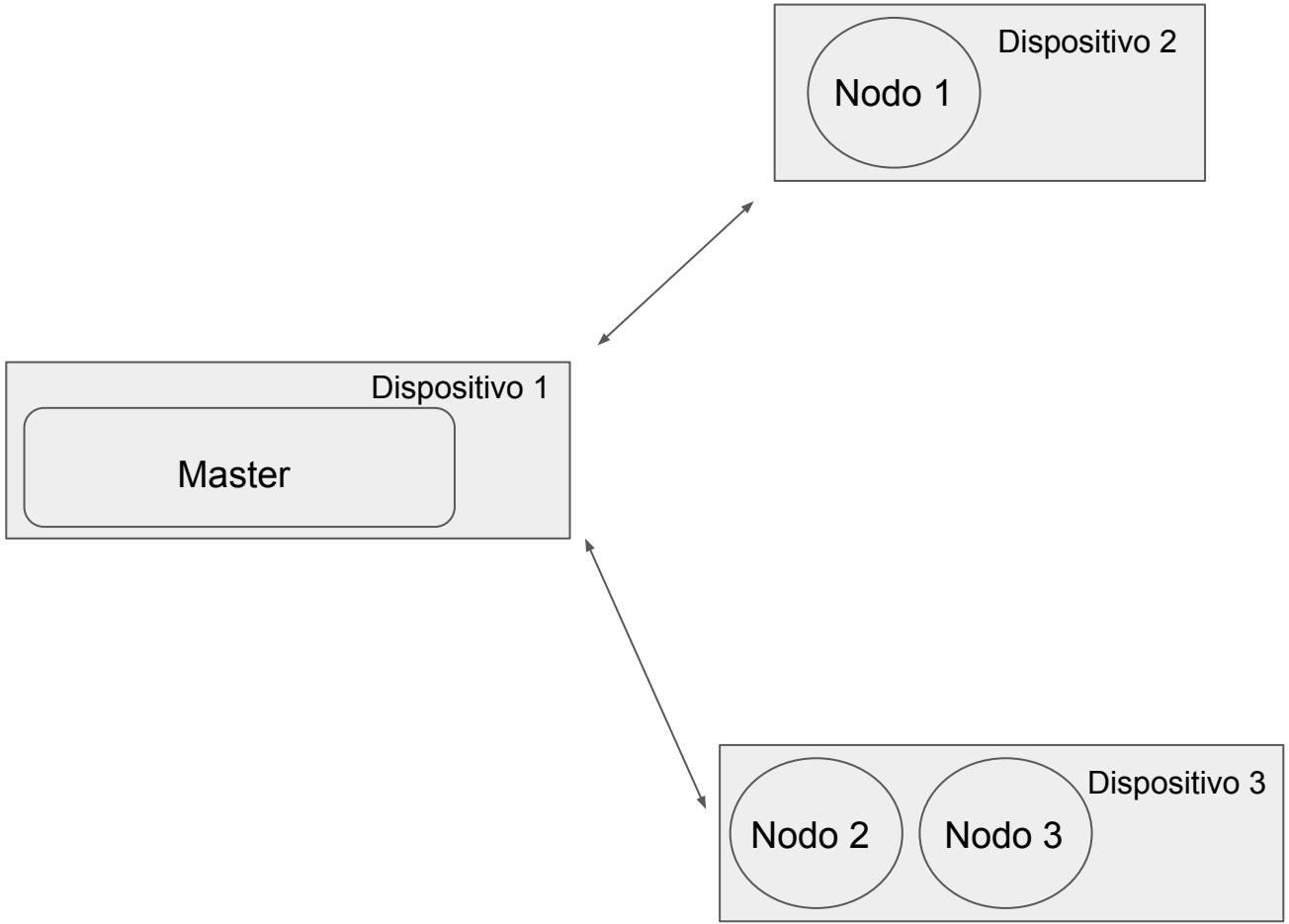
Parâmetros

Tópicos

Serviços

Bags

Workspace


Dispositivo 2
Nodo 1
Dispositivo 1
Master
Dispositivo 3
Nodo 2
Nodo 3

Master

- **Nodo**

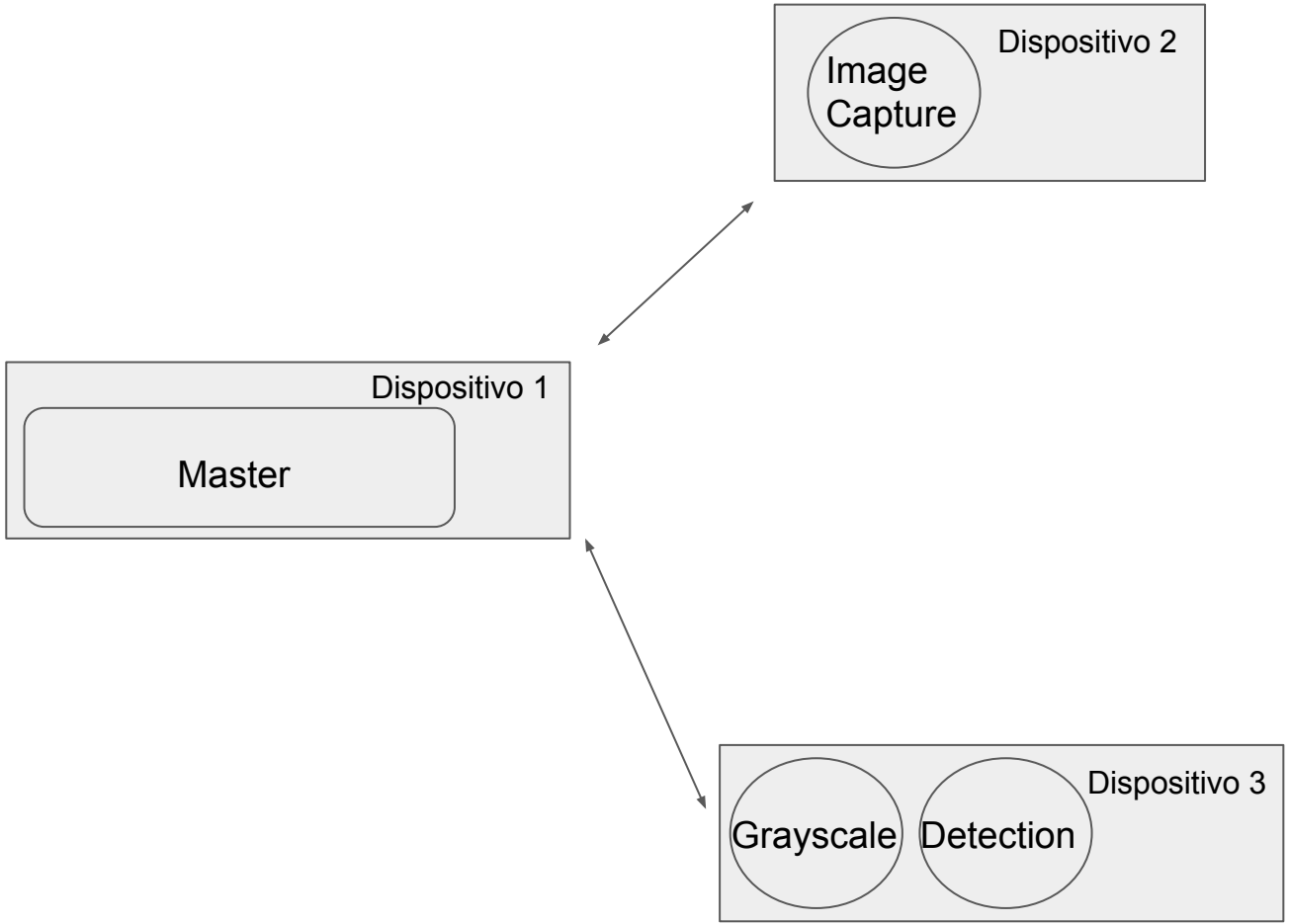
Parâmetros

Tópicos

Serviços

Bags

Workspace


Dispositivo 2
Image Capture
Dispositivo 1
Master
Dispositivo 3
Grayscale
Detection

Master

Nodo

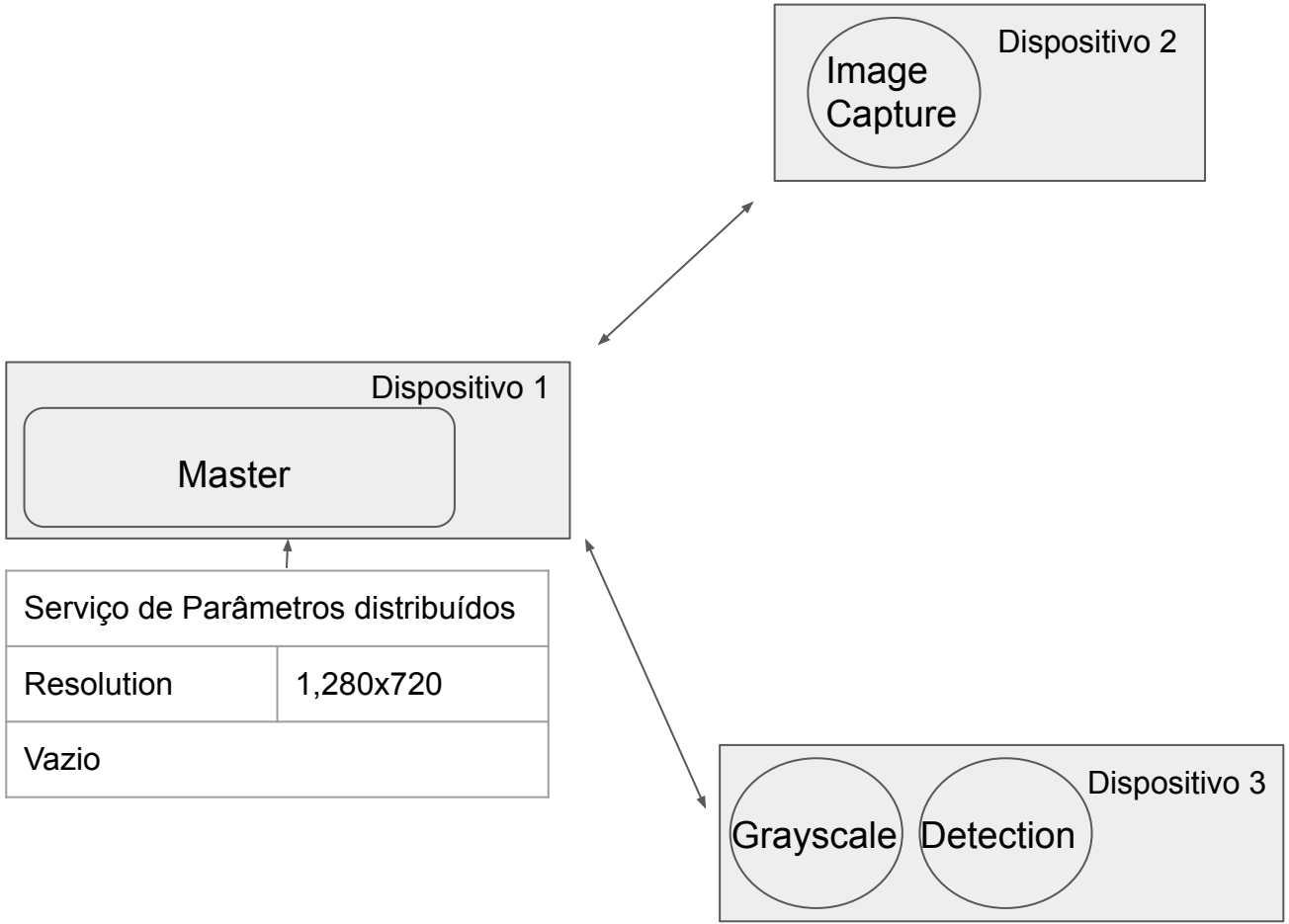
- **Parâmetros**

Tópicos

Serviços

Bags

Workspace


Dispositivo 2
Image Capture
Dispositivo 1
Master
Serviço de Parâmetros distribuídos
Resolution
1,280x720
Vazio
Dispositivo 3
Grayscale
Detection

Master

Nodo

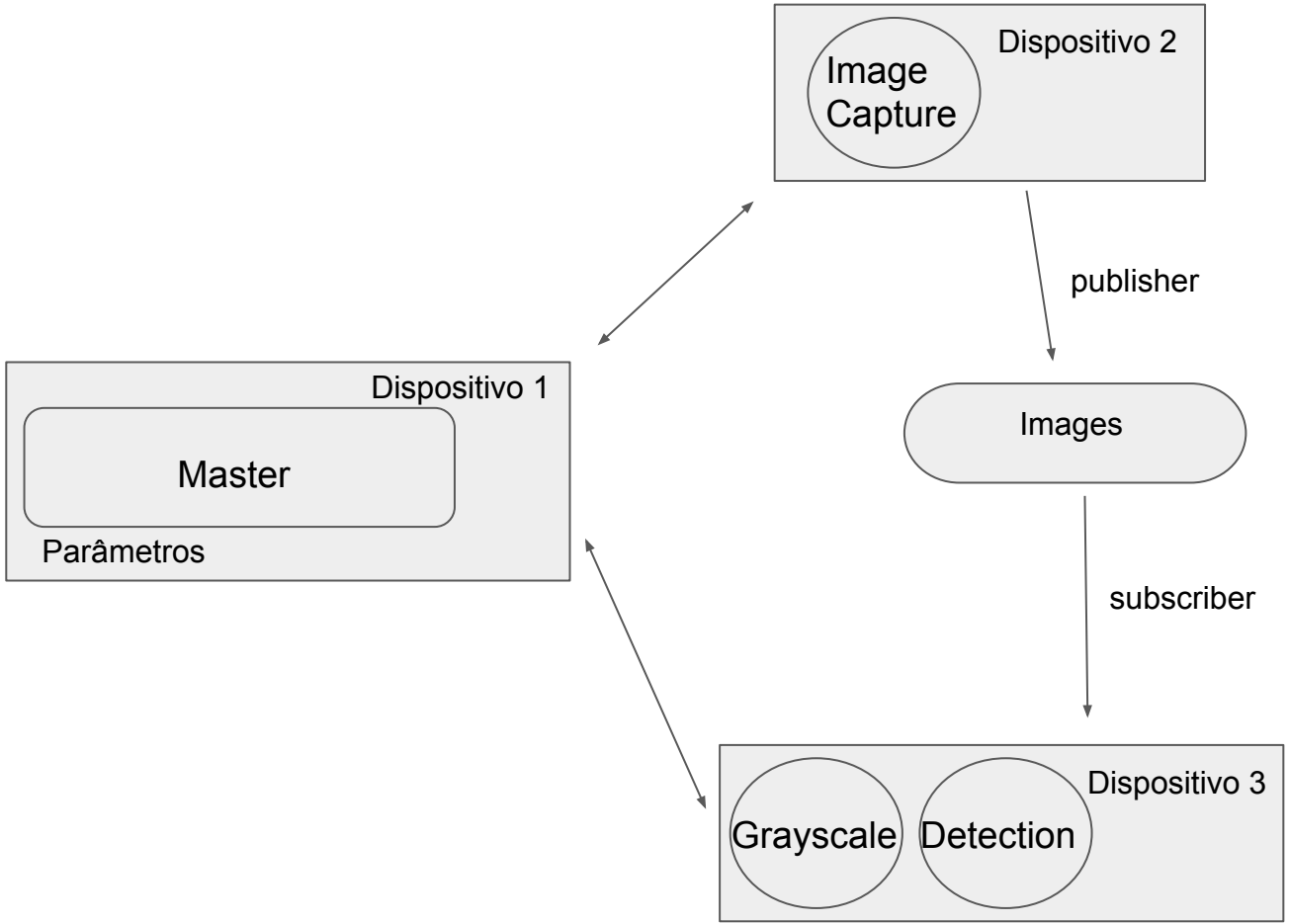
Parâmetros

- **Tópicos**

Serviços

Bags

Workspace


Dispositivo 2
Image Capture
publisher
Images
subscriber
Dispositivo 1
Master
Parâmetros
Dispositivo 3
Grayscale
Detection

Master

Nodo

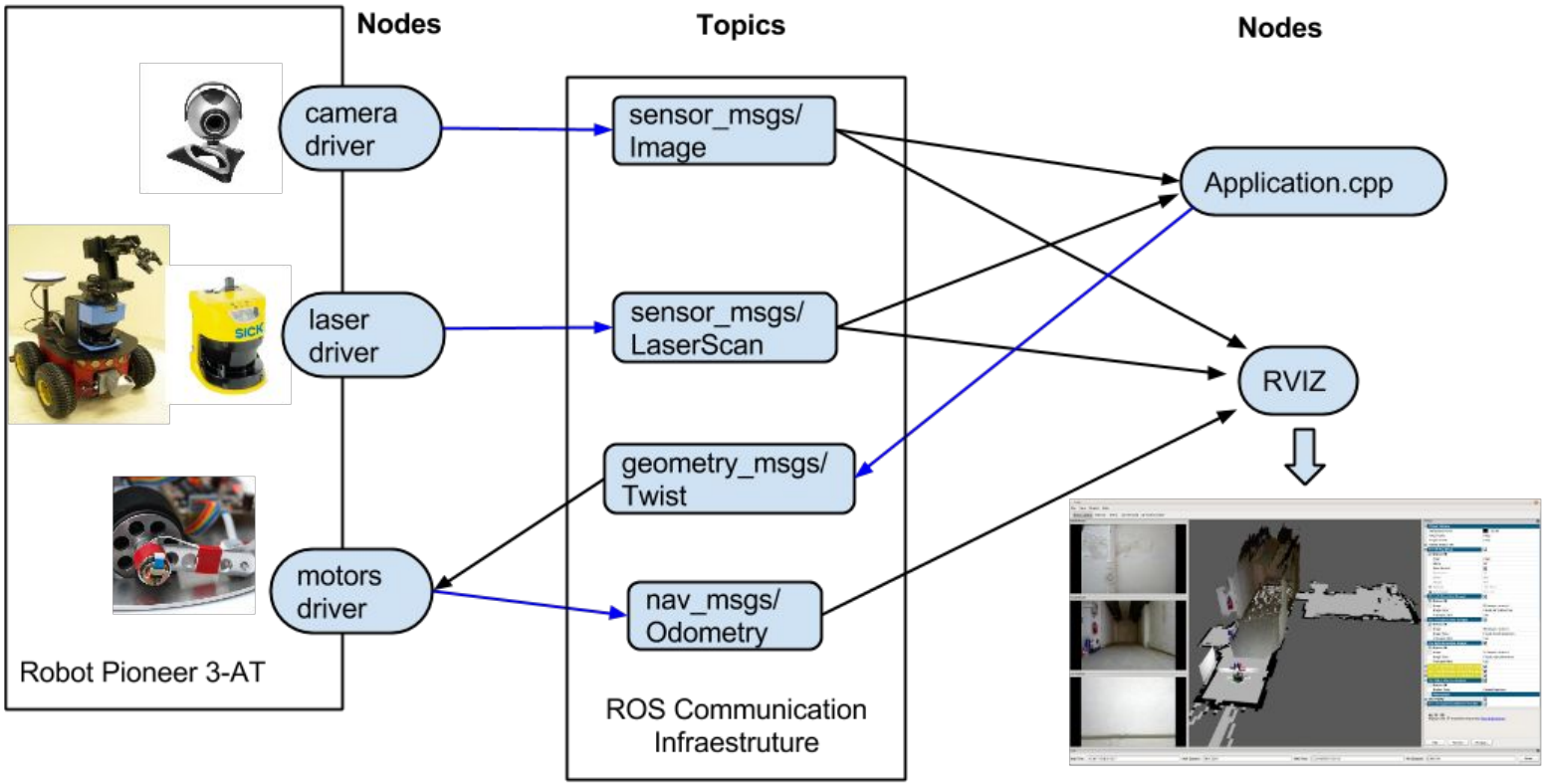
Parâmetros

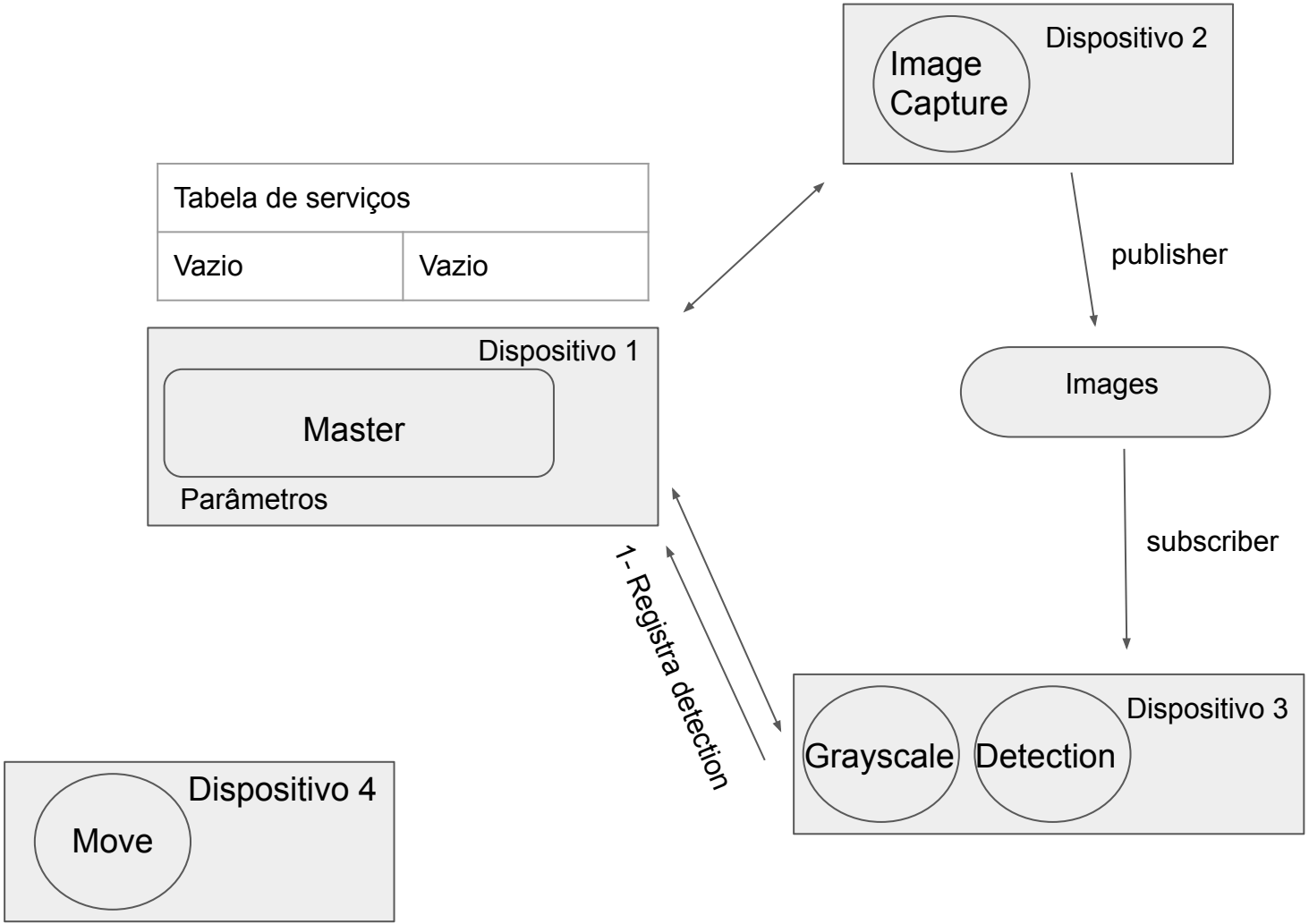
Tópicos

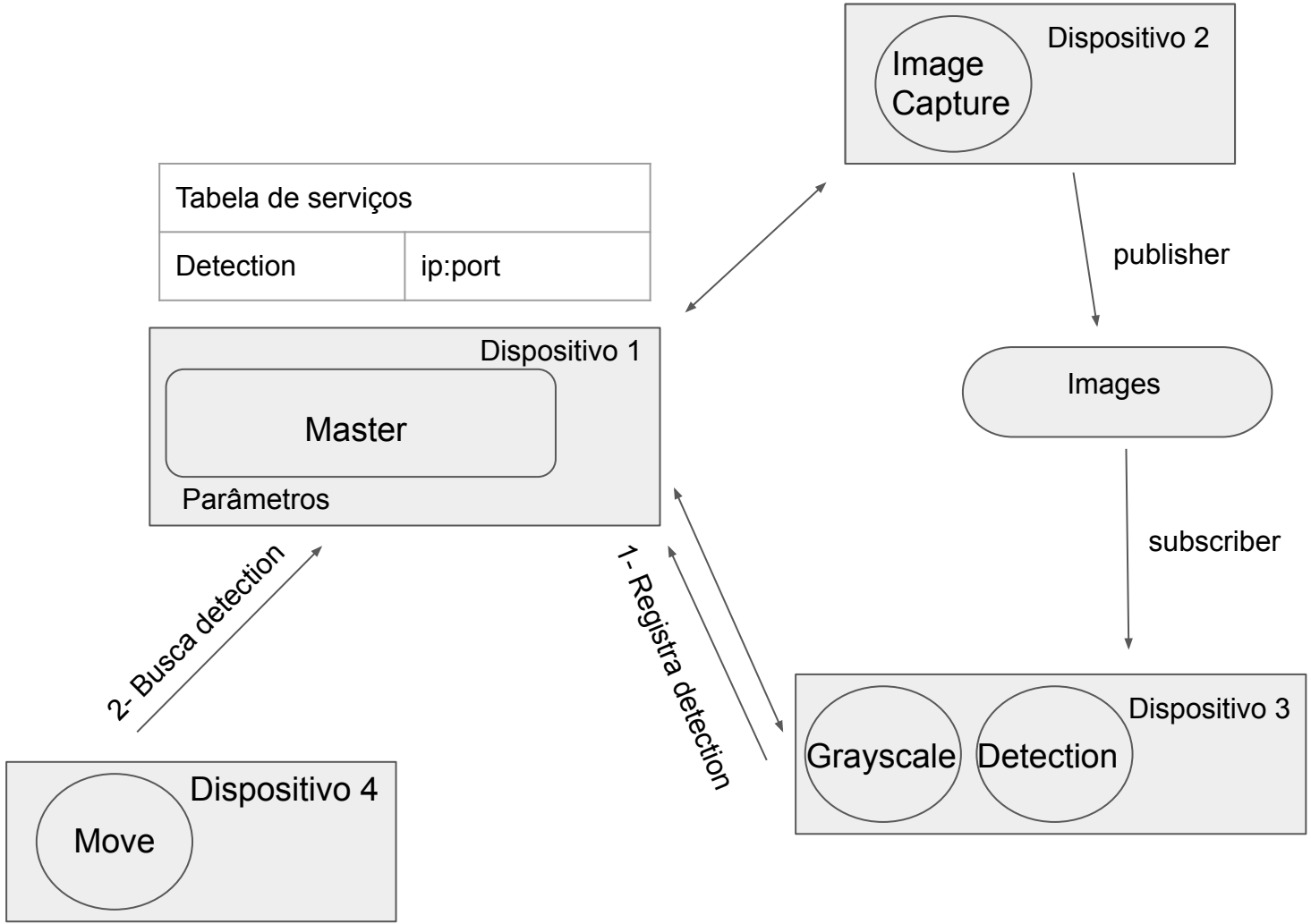
- **Serviços**

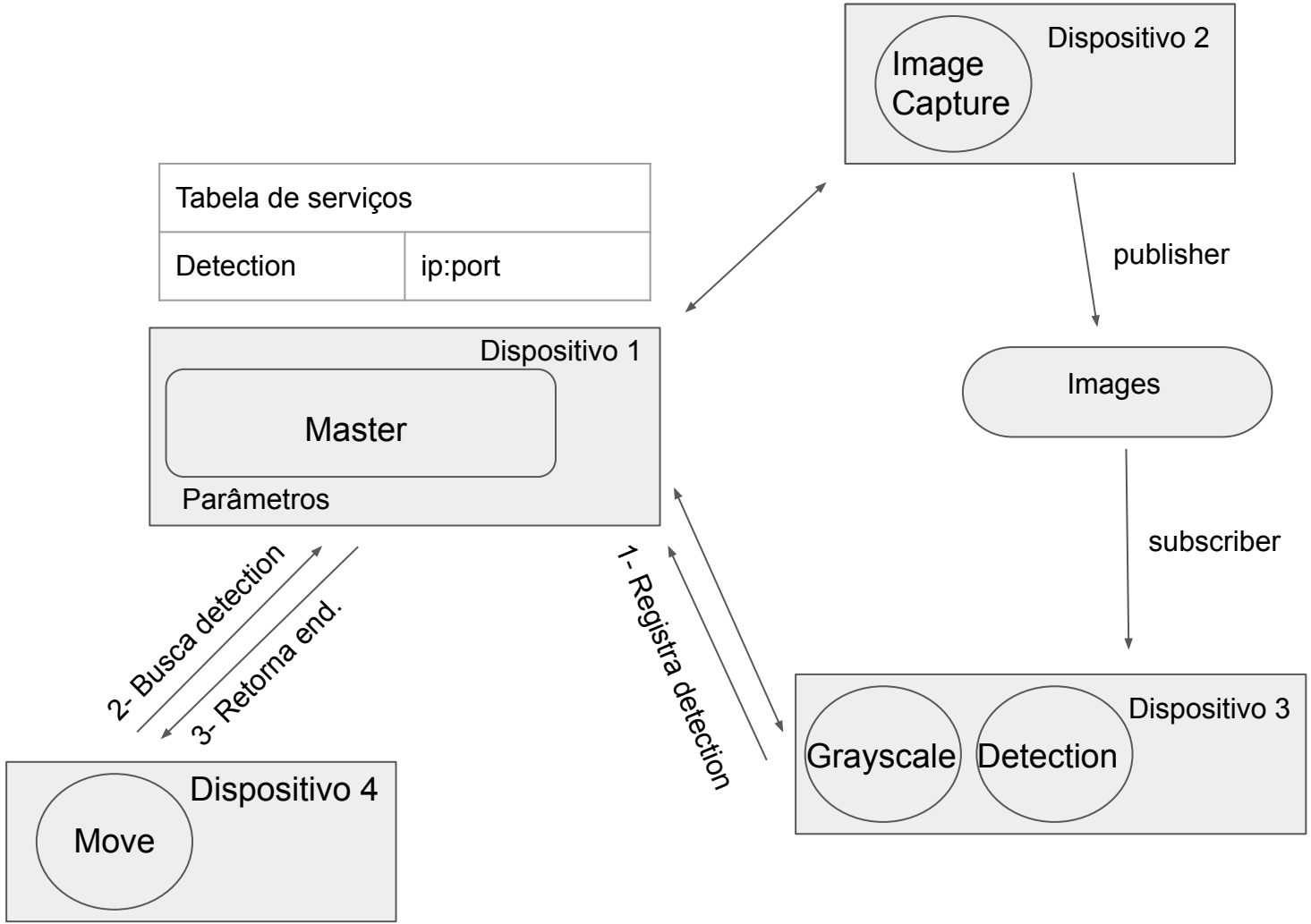
Bags

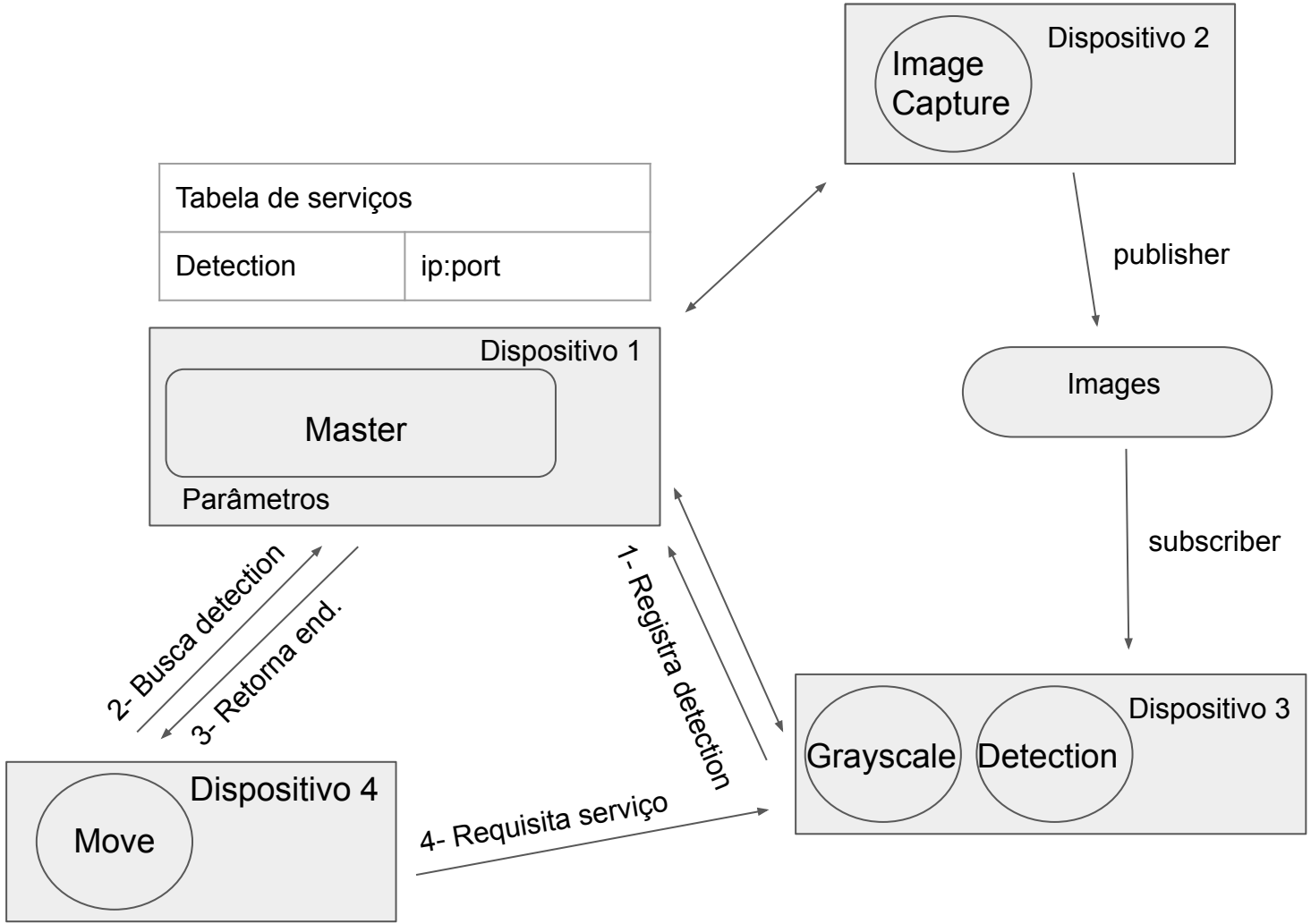
Workspace

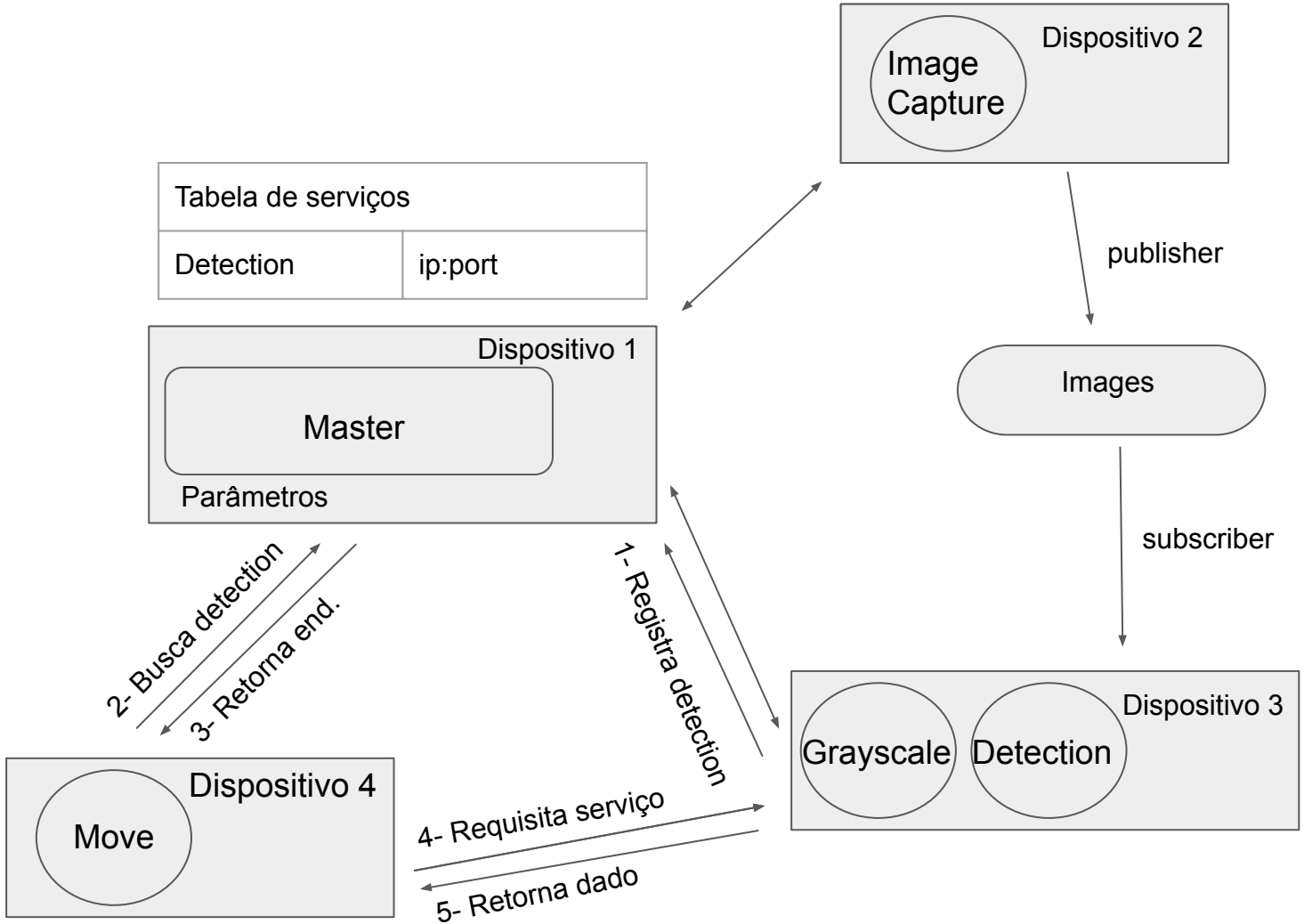

Nodes
Topics
Nodes
camera driver
laser driver
motors driver
sensor_msgs/ Image
sensor_msgs/ LaserScan
geometry_msgs/ Twist
nav_msgs/ Odometry
Application.cpp
RVIZ
Robot Pioneer 3-AT
ROS Communication Infraestruture


Dispositivo 2
Image Capture
Tabela de serviços
Vazio
Vazio
publisher
Dispositivo 1
Master
Parâmetros
Images
subscriber
1- Registra detection
Dispositivo 3
Grayscale
Detection
Dispositivo 4
Move


Dispositivo 2
Image Capture
Tabela de serviços
Detection
ip:port
publisher
Dispositivo 1
Master
Parâmetros
Images
subscriber
2- Busca detection
1- Registra detection
Dispositivo 3
Grayscale
Detection
Dispositivo 4
Move


Dispositivo 2
Image Capture
Tabela de serviços
Detection
ip:port
publisher
Dispositivo 1
Master
Parâmetros
Images
subscriber
2- Busca detection
3- Retorna end.
1- Registra detection
Dispositivo 3
Grayscale
Detection
Dispositivo 4
Move


Dispositivo 2
Image Capture
Tabela de serviços
Detection
ip:port
publisher
Dispositivo 1
Master
Parâmetros
Images
subscriber
2- Busca detection
3- Retorna end.
1- Registra detection
Grayscale
Detection
Dispositivo 3
Dispositivo 4
Move
4- Requisita serviço


Dispositivo 2
Image Capture
Tabela de serviços
Detection
ip:port
publisher
Dispositivo 1
Master
Parâmetros
Images
subscriber
2- Busca detection
3- Retorna end.
1- Registra detection
Grayscale
Detection
Dispositivo 3
Dispositivo 4
Move
4- Requisita serviço
5- Retorna dado

Master

Nodo

Parâmetros

Tópicos

Serviços

- **Bags**

Workspace

Master

Nodo

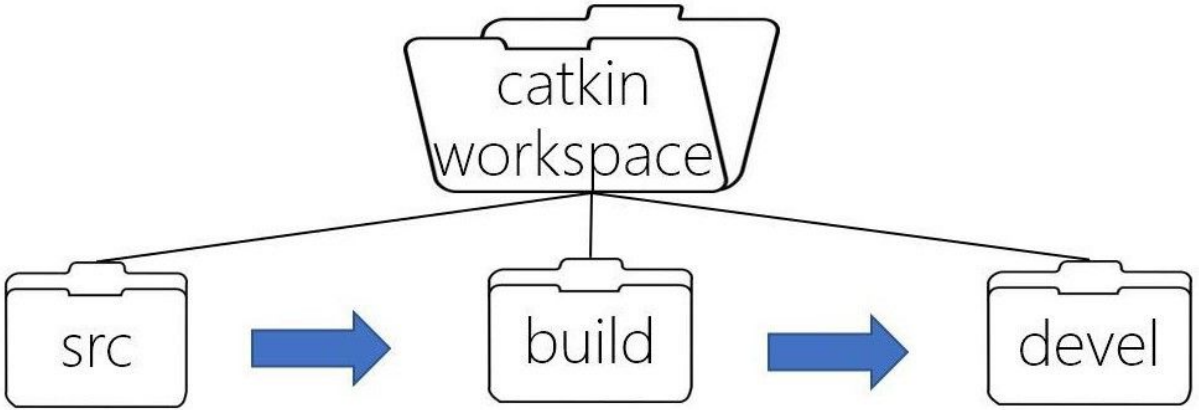
Parâmetros

Tópicos

Serviços

Bags

- **Workspace**



Instalando o ROS

Esta instalação se refere à versão ROS Noetic, compatível com Ubuntu 20.04 (Focal), em caso de outra versão desejada, por favor, entre em http://wiki.ros.org/noetic/Installation

```
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://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

```
sudo apt-get update
```

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

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

```
source ~/.bashrc
```

Inicie o Master!

$ roscore

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

$ rosnode list

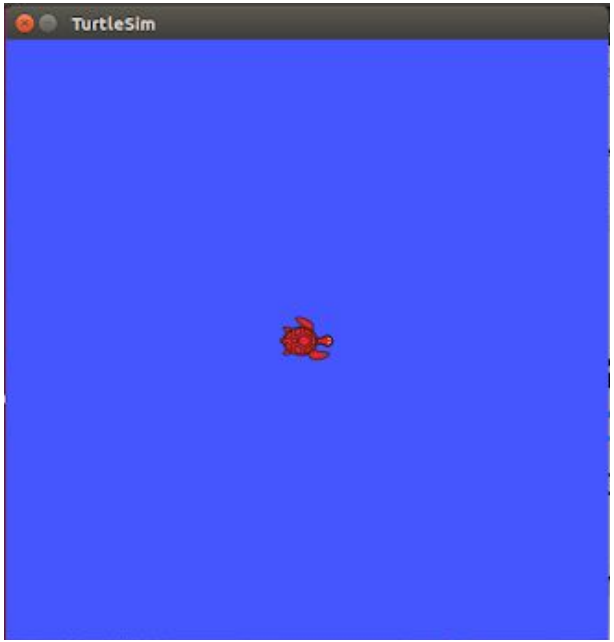
Brincando com o Turtlesim

Execute o turtlesim:

$ rosrun turtlesim turtlesim_node

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

$ rosnode list



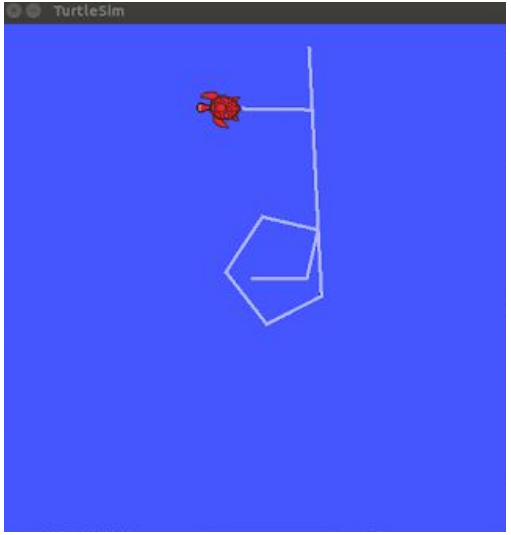
Controlando a turtle

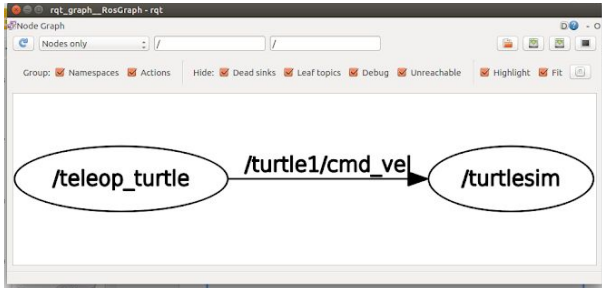
Agora controle a tartaruga! Em um novo terminal digite:

$ rosrun turtlesim turtle_teleop_key

É possível também ver as conexões dos nodos do ROS digitando:

$ rosrun rqt_graph rqt_graph





Tópicos

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

Publicando mensagens

- Comando e argumentos:

  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]'

Tópicos

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

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

Convocando as tartarugas ninjas!

Descubra os argumentos do serviço spawn:

$ rosservice type /spawn | rossrv show

Mate a tartaruga existente :(

$ rosservice call /kill turtle1

Convocando as tartarugas ninjas!

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

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]

Parâmetros

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

Revisão

roscore - Inicia o master;

rosrun - Executa um nodo;

rosnode - Funções relacionadas a nodos;

rosservice - Funções relacionadas a serviços;

rostopic - Funções relacionadas a tópicos;

rosparam - Funções relacionadas a parâmetros;

Workspace

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

Criando o workspace:

$ catkin_init_workspace

Compilando o workspace:

$ cd ..

$ catkin_make

Pacotes

Acesse a pasta:

$ cd ~/catkin_ws/src

Execute o script para a criação:

#catkin_create_pkg <Nome do Pacote> [Dependências]

$ catkin_create_pkg tutorial std_msgs rospy roscpp

Para compilar:

cd ~/catkin_ws && catkin_make

Bashrc

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

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/tutorial/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-->


</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 launchs 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 <sstream>

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;
}
```

Executando

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

Executando

Para executar os nodos no pacote.

Terminal 1:

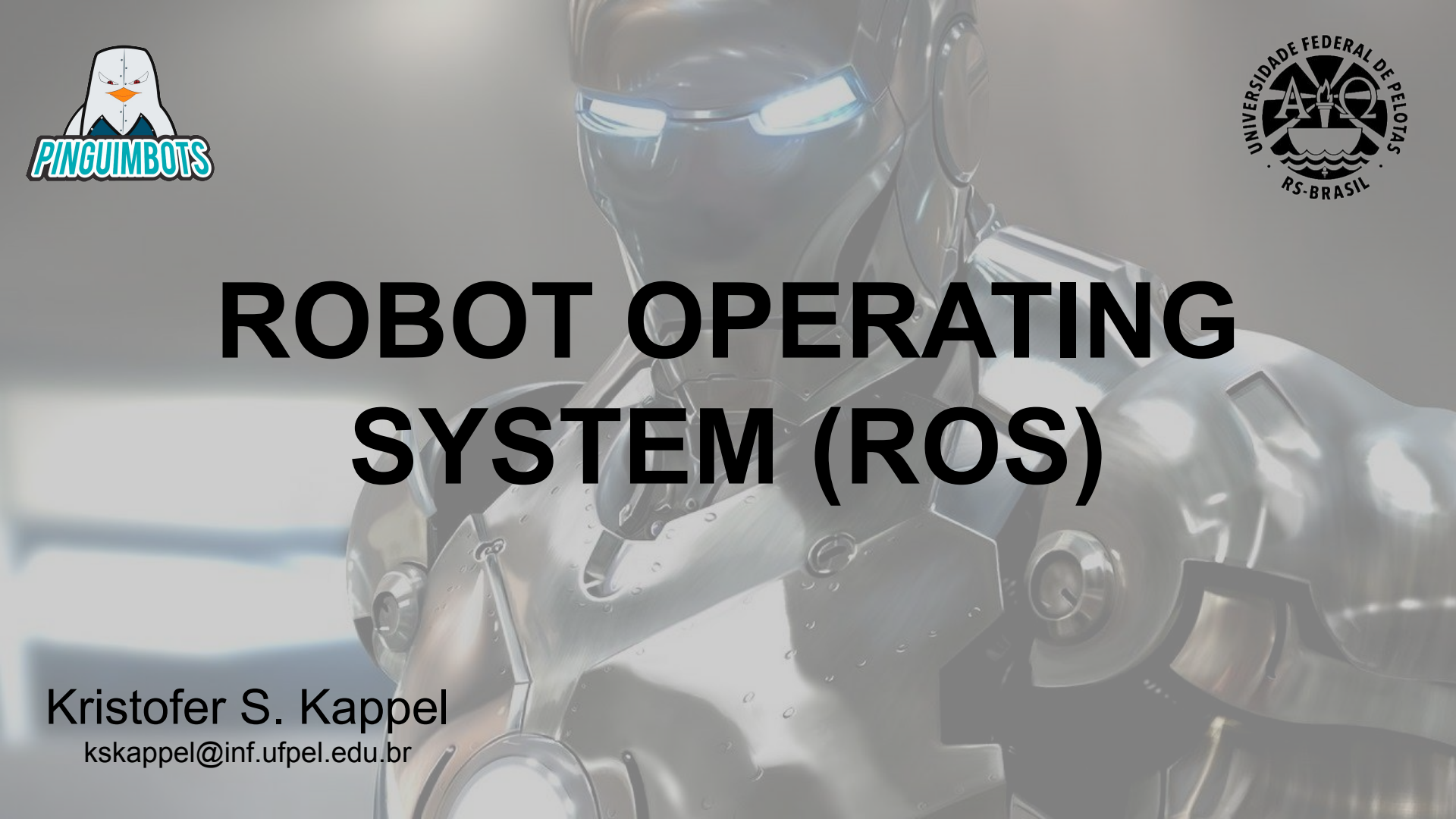
$ roscore

Terminal 2:

$ rosrun tutorial talker_node

Terminal 3:

$ rosrun tutorial listener_node



ROBOT OPERATING SYSTEM (ROS)

Kristofer S. Kappel
kskappel@inf.ufpel.edu.br