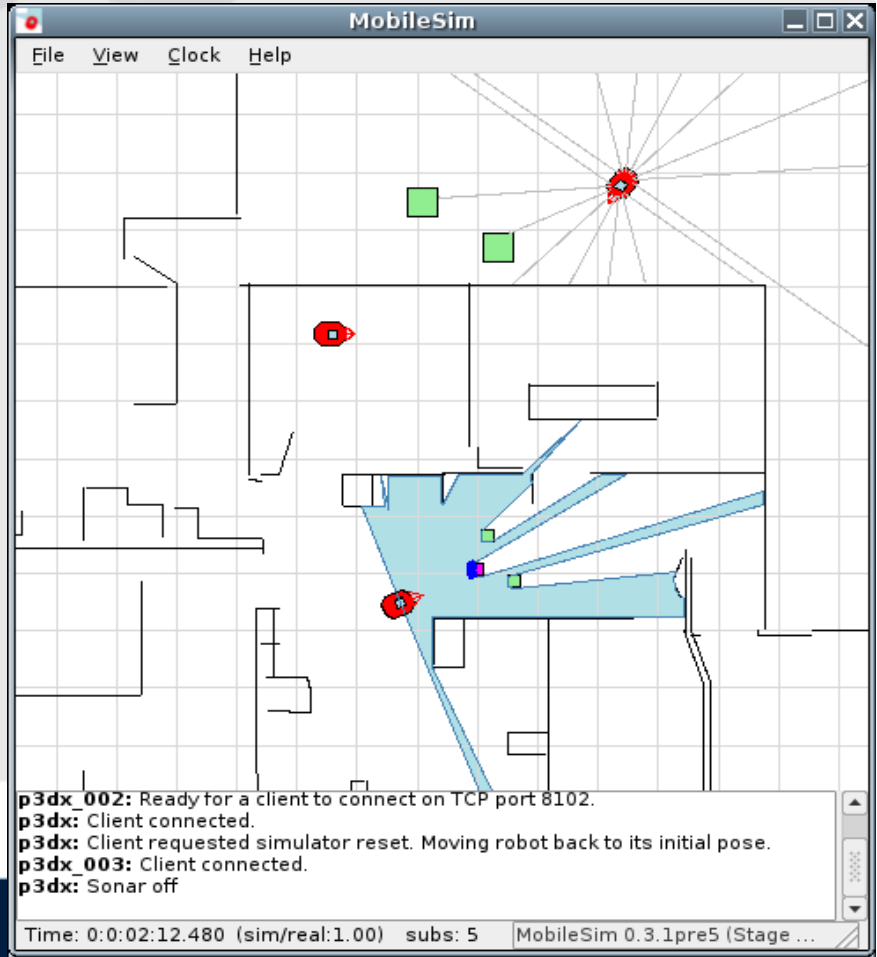
Universidade Federal de Itajubá

Aria e Rosaria

Aria e Rosaria

Os robôs Pioneer 3DX e AmigoBot são robôs desenvolvidos pela empresa Adept MobileRobot que disponibiliza uma biblioteca open source, chamada Aria, para facilitar na criação de novas aplicações.



Aria e Rosaria

Ao trabalhar com o Rosaria, o programador desfruta da facilidade dos dois mundos:

- Através do Aria, a conexão dos robôs é feita com poucas linhas de código e existem várias classes já criadas disponíveis nesta biblioteca.
- Através do Rosaria, é possível visualizar as informações em cada tópico e enviar comandos através das ferramentas do ROS. Além de todas as funções que o ROS oferece.

http://www.ai.rug.nl/vakinformatie/pas/content/Aria-manual/classes.html

Aria e Rosaria

Abra o simulador Mobilesim

Escolha o modelo do robô e clique em 'no map' ou escolha um mapa disponível.

Abra um terminal: *$ roscore*

Abra outro terminal: *$ rosrun rosaria RosAria*

$ rostopic list

$ rostopic pub /RosAria/cmd_vel geometry_msgs/Twist '[1.0, 0.0, 0.0]' '[0.0, 0.0, 2.0]'

$ rosservice list

$ rosservice call /disable_motors

Universidade Federal de Itajubá

Conceitos

Conceitos

Pacotes (Packages)

Pilhas vs Meta-pacotes (Stacks vs Meta-packages)

Nós (Nodes)

Mestre (Master)

Launch

Tópicos (Topics)

Mensagens (Messages)

Serviços (Services)

Bags

Packages

Os pacotes são a principal unidade de organização no ROS.

Agrupam arquivos de mesma natureza:

Nome_do_pacote

- includes…… cabeçalhos (.h)
- src (source)…… códigos-fonte (.cpp)
  - camera_node.cpp
  - processamento_imagem.cpp
- manifest.xml…… informações sobre o pacote
- Cmakelists.txt…….. configuração Cmake
- launch, msg, srv etc

Packages

Para criar um pacote, abra o terminal e execute os seguintes comandos:

- *$ cd*
- *$ cd catkin_ws/src/*
- *$ catkin_create_pkg* ***curso_de_ros*** *std_msgs rospy roscpp*
- *$ cd ..*
- *$ catkin_make*

Stacks vs Meta-Packages

O conceito de stack foi utilizado até o ROS Fuerte (compilador *rosbuild*) e nas versões posteriores (*catkin*) foi implementado os meta-packages.

A definição básica dos dois se manteve a mesma:

- os meta-packages são pacotes que agrupam outros pacotes que estão relacionados entre si.

Nodes

- Nós são processos que executam algum tipo de computação, podendo ser considerados como executáveis.

- Por exemplo: emissor e receptor.

Nodes

- Iremos criar dois nós:
  - emissor_node
  - receptor_node
- Na pasta: */catkin_ws/src/curso_de_ros/src*
  - Crie um arquivo novo chamado *emissor.cpp*

Nodes

- Dentro do arquivo coloque as seguintes linhas:

```
#include "ros/ros.h"

int main(int argc, char
**argv)
{
  ros::init(argc, argv,
"emissor");

  ros::NodeHandle n;

  ros::Rate loop_rate(10);

  while (ros::ok())
  {
    ros::spinOnce();

    loop_rate.sleep();
  }
  return 0;
}
```

Nodes

- Na mesma pasta crie um arquivo chamado *receptor.cpp* e coloque o código abaixo:

```
#include "ros/ros.h"

int main(int argc, char
**argv)
{
  ros::init(argc, argv,
"receptor");
  ros::NodeHandle n;

  ros::spin();

  return 0;
}
```

Nodes

- Na pasta */catkin_ws/src/curso_de_ros* abra o arquivo *CMakeLists.txt* e faça as seguintes modificações:

```
add_executable(emissor src/emissor.cpp)
target_link_libraries(emissor ${catkin_LIBRARIES})
add_dependencies(emissor curso_de_ros_generate_messages_cpp)

add_executable(receptor src/receptor.cpp)
target_link_libraries(receptor ${catkin_LIBRARIES})
add_dependencies(receptor curso_de_ros_generate_messages_cpp)
```

Nodes

- Para compilar o projeto:
  - *$ roscd*
  - *cd ..*
  - *$ catkin_make*

Master

Sempre deverá ser inicializado primeiramente.

É um nó que provê serviços de registro e consulta de nomes de outros nós, de tópicos e de serviços.

Sempre que um nó é iniciado, ele se registra com o mestre.

O nó que subscreve um tópico faz uma consulta ao mestre e estabelece uma conexão direta com este.

Nodes

- Pimeiramente deve-se executar o master:
  - *$ roscore*
- Em outros terminais:
  - *$ rosrun curso_de_ros emissor*
  - *$ rosrun curso_de_ros receptor*

Nodes

- Pimeiramente deve-se executar o master:
  - *$ roscore*
- Em outros terminais:
  - *$ rosrun curso_de_ros emissor*
  - *$ rosrun curso_de_ros receptor*
  - *$ rqt_graph*

Nodes

- *$ rosnode -h*
- *$ rosnode list*
- *$ rosnode info /nome_do_nó*

Launch

- Um arquivo *.launch* possibilita executar vários nós através de apenas um comando.
- Além dos nós desejados, ele também executa o *master.*
- Portanto, ao utilizar um launch, torna-se desnecessário executar o *roscore*.

Launch

- Abra um terminal:
  - *$ cd ~/catkin_ws/src/curso_de_ros*
  - *$ mkdir launch*
  - *$ cd launch*
  - *$ gedit comunicacao.launch*
  - Coloque o seguinte código:

```
<launch>
    <node pkg="curso_de_ros" name="emissor" type="emissor"/>
    <node pkg="curso_de_ros" name="receptor" type="receptor"/>
</launch>
```

Messages

Os nós se comunicam entre si através de mensagens.

Sendo cada uma com um tipo diferente de informação:

- integer;
- boolean;
- array.

E também um conjunto de tipos primitivos (struct):

- geometry_msgs/Pose2D
- http://docs.ros.org/api/geometry_msgs/html/msg/Pose2D.html

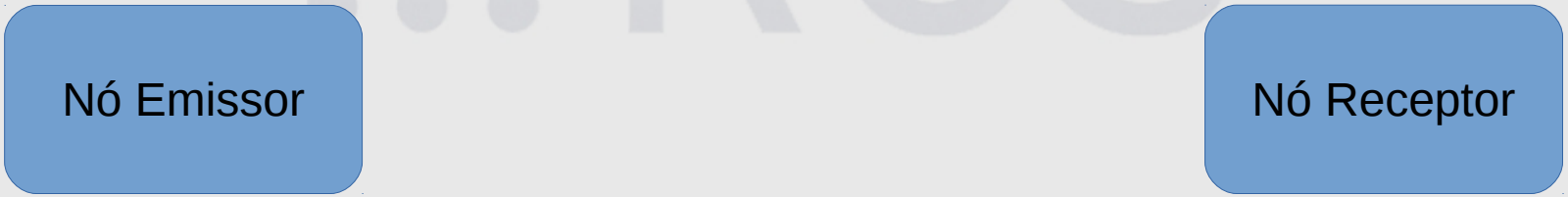
Pode-se também criar sua própria mensagem (será abordado mais a frente).

Topics

- Os nós trocam informações através dos tópicos, que podem ser considerados como um barramento de dados.
- Os nós que colocam dados em um certo tópico são chamados de publisher.
- Enquanto os nós que fazem a leitura dos dados presentes no tópico, são os subscriber.
- Em um mesmo nó podemos ter tanto publishers quanto subscribers.
- OBS: o tipo do tópico deverá ser igual ao tipo da mensagem.

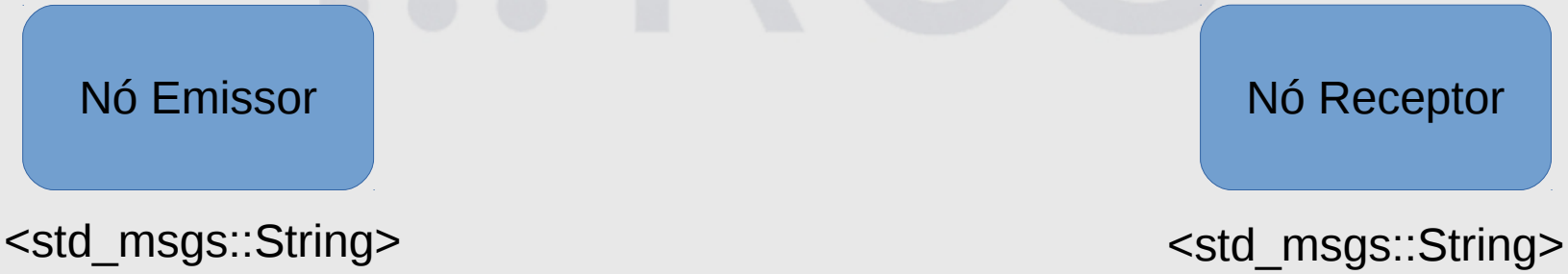
Topics

- Exemplo emissor-receptor



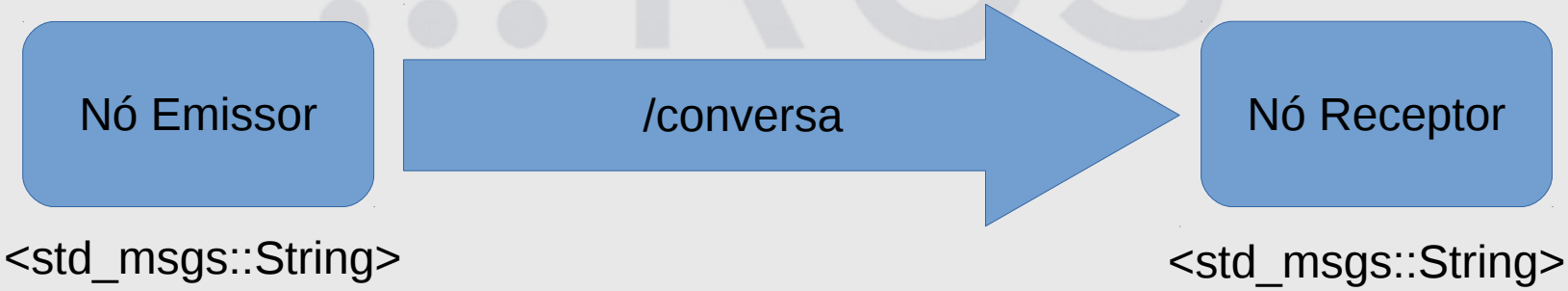
Topics

- Exemplo emissor-receptor



Topics

- Exemplo emissor-receptor



Topics

- Robótica