

Universidade Tecnológica Federal do Paraná (UTFPR)
Disciplina: CPGEI/PPGCA - Robótica Móvel

Introdução à Robótica Móvel

através do Robot Operating System (ROS)

Prof. André Schneider de Oliveira
Prof. João Alberto Fabro

Robôs móveis com ROS



Robot Operating System (ROS)

- *Framework* que atua sobre o Linux para a padronização de mensagens em sistemas robóticos

www.ros.org

- A padronização permite o compartilhamento de soluções
- Existem diferentes versões do framework, cada uma voltada à uma versão do Linux Ubuntu
 - ROS Indigo Igloo → Ubuntu 14.04.* LTS → abril de 2019
 - ROS Jade Turtle → Ubuntu 15.04
 - ROS Kinetic Kame → Ubuntu 16.04.* LTS → abril de 2021

Robot Operating System (ROS)

- Possui nós para diferentes finalidades
 - interface com sensores e atuadores
 - comunicação entre dispositivos
 - navegação e mapeamento
 - mapeamento tridimensional
 - experimentação virtual
 - entre outros....

Requisitos para as atividades práticas

- Instalação do ambiente de desenvolvimento
 1. máquina virtual VMBox **(pode ficar lento!)**
 2. nativamente em uma partição linux **(requer cuidado!)**
- Conhecimentos básicos de Linux
- Conhecimentos em Programação C ou Python
 - funções, bibliotecas e estruturas compostas

Site da disciplina

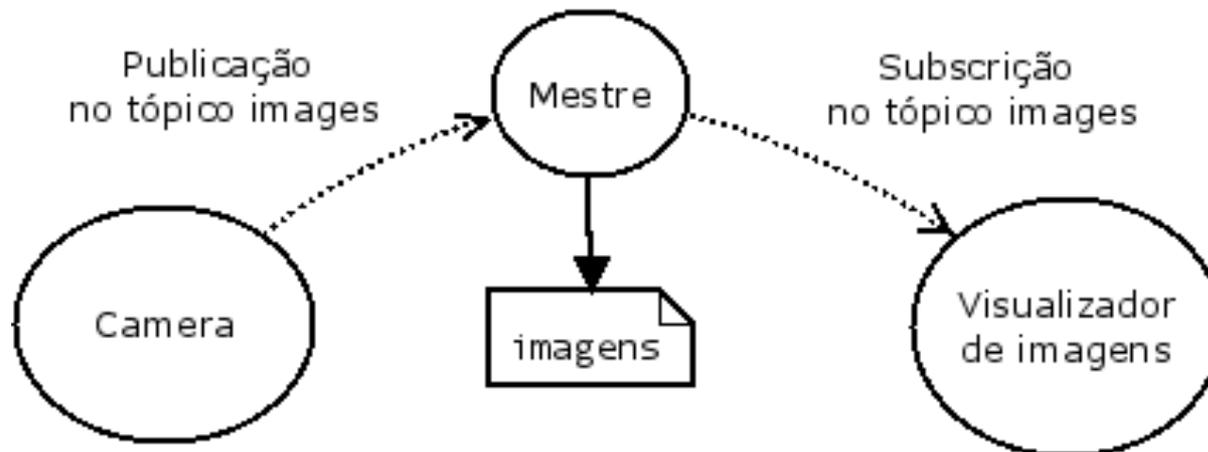
<http://dainf.ct.utfpr.edu.br/~andre/>

Softwares:

- *  Instalação Robot Operating System (ROS)
- * Configuração dos computadores da B204
- *  Ambiente de experimentação virtual com plugin ROS compilado
- *  Ambiente matematico compativel com o ROS
- *  Máquina Virtual (VMBox) Lubuntu+Ros_Kinetic
- *  Ambiente STAGE para as tarefas da disciplina

Robot Operating System (ROS)

- O ROS cria uma estrutura de comunicação entre nós (softwares) de diferentes origens e finalidades
- A estrutura do ROS é denominada de mestre (*master*) ROS



Master ROS

- Inicializado pelo comando

\$ roscore

```
turtlebot@turtlebot-X200CA:~$ roscore
... logging to /home/turtlebot/.ros/log/6ef6185c-9127-11e4-83da-0c84dc11754b/ros
launch-turtlebot-X200CA-9168.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.0.8:45853/
ros_comm version 1.11.9

SUMMARY
=====

PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.9

NODES

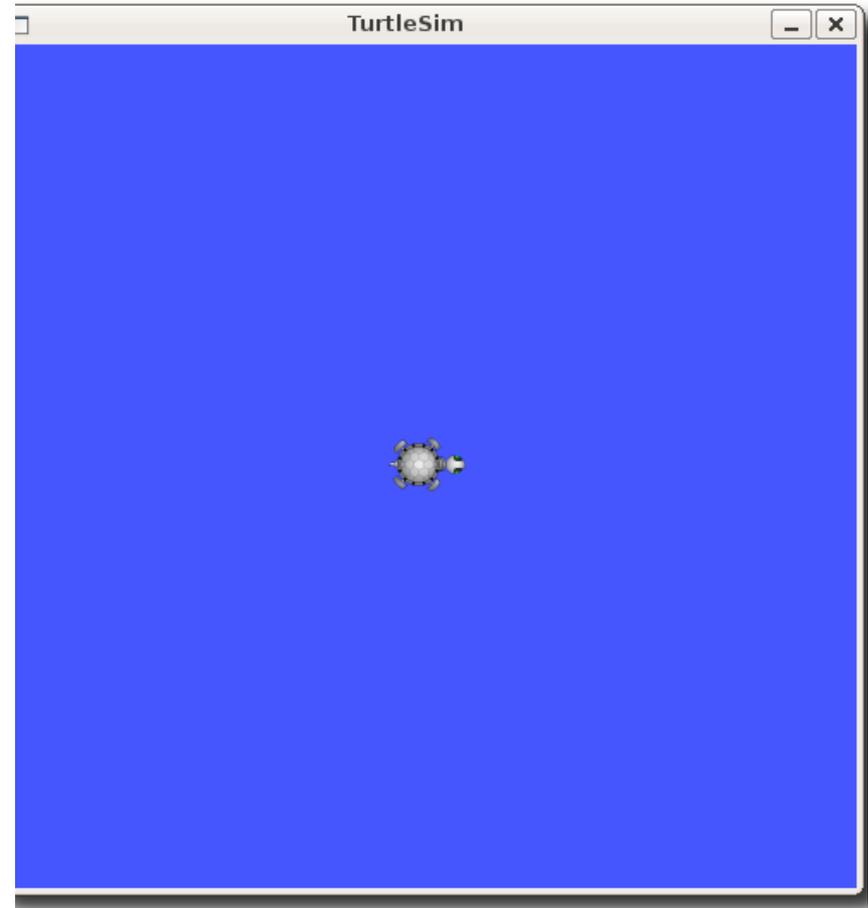
auto-starting new master
process[master]: started with pid [9180]
ROS_MASTER_URI=http://192.168.0.8:11311/

setting /run_id to 6ef6185c-9127-11e4-83da-0c84dc11754b
process[rosout-1]: started with pid [9193]
started core service [/rosout]
```

TurtleSim

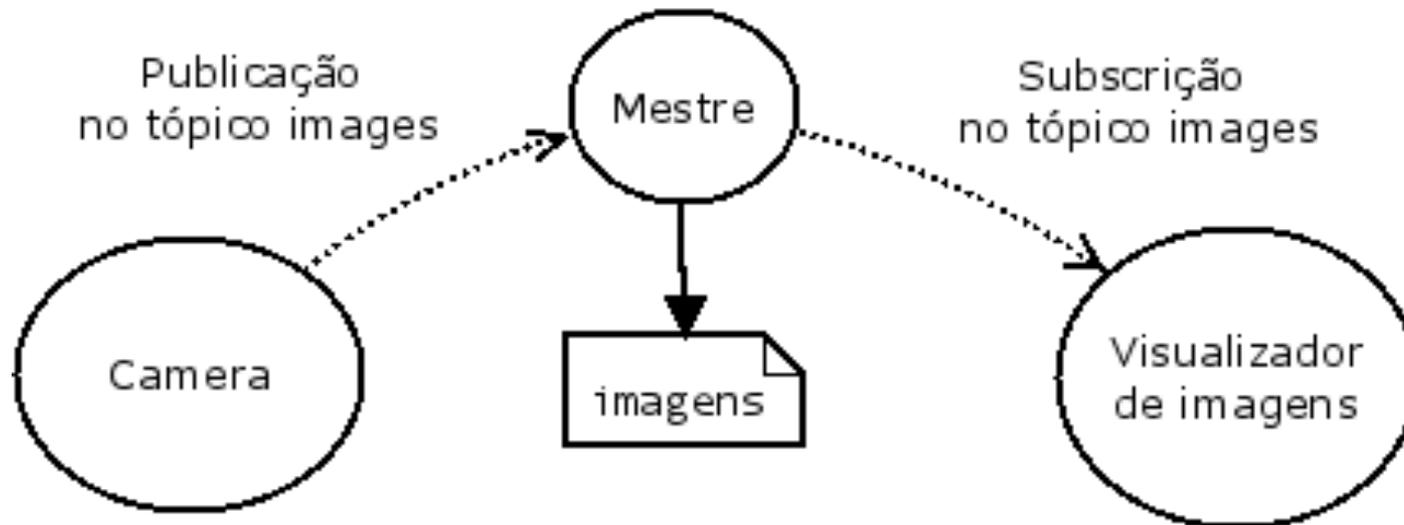
- Simulador integrado do ROS
- Executado pelo comando

```
$ roslaunch turtlesim turtlesim_node
```



Robot Operating System (ROS)

- O ROS cria uma estrutura de comunicação de softwares por lista de mensagens



Mensagens

- Existe uma grande quantidade de mensagens padronizadas no ROS
- principais bibliotecas de mensagens
 - ***std_msgs*** → mensagens primitivas → int, float, string, time
 - ***common_msgs*** → pacote com os principais tipos de mensagens
 - ***geometry_msgs*** → primitivas geométricas → acceleration, pose2D, twist
 - ***sensor_msgs*** → sensores → image, imu, pointcloud, laserscan
 - ***nav_msgs*** → navegação → gridcells, occupancygrid, path

Exemplo de Mensagens

Type: geometry_msgs/Twist

```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

Type: nav_msgs/Odometry

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
    float64[36] covariance
  geometry_msgs/TwistWithCovariance twist
    geometry_msgs/Twist twist
      geometry_msgs/Vector3 linear
        float64 x
        float64 y
        float64 z
      geometry_msgs/Vector3 angular
        float64 x
        float64 y
        float64 z
      float64[36] covariance
```

Mensagens

- Listar mensagens disponíveis

```
$ rosmg list
```

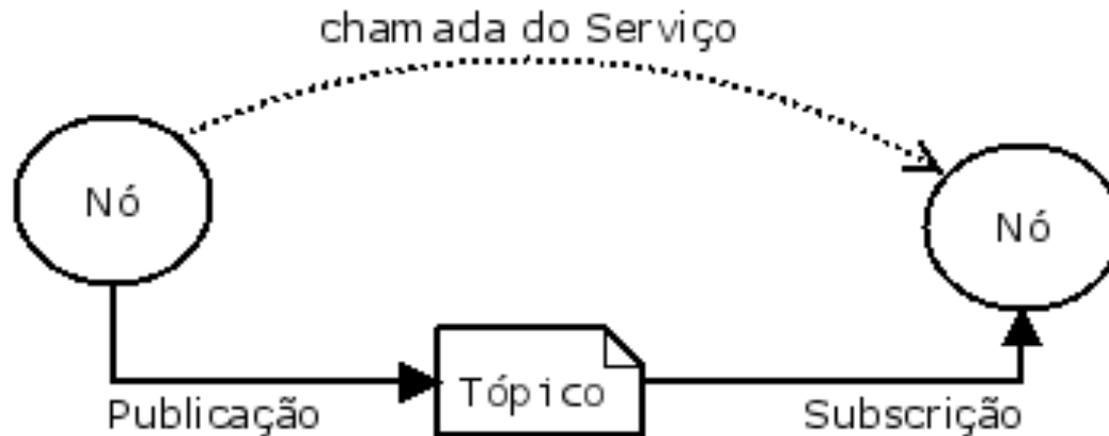
- Ver o conteúdo de uma mensagem

```
$ rosmg show [tipo_da_msg]
```

```
$ rosmg show geometry_msg/Twist
```

Tópicos

- A lista de mensagens do ROS é subdividida em tópicos, ou espaços para mensagens
- Os tópicos armazenam as mensagens que estão trafegando pelo mestre ROS ("buffer")



Tópicos

- Os tópicos ativos do ROS podem ser visualizados por

```
$ rostopic list
```

- O conteúdo do tópico é visto por

```
$ rostopic echo [nome_do_topico]
```

- As informações do tópico (tipo de msg) pode ser acessada pelo comando

```
$ rostopic info [nome_do_topico]
```

Nós

- Os Nós são os programas ativos no ROS que subscrevem e publicam no tópicos
- Visualizar os nós ativos

```
$ rostopic list
```

- Um nó é executado por

```
$ roslaunch [nome_do_pacote] [nome_do_nó]
```

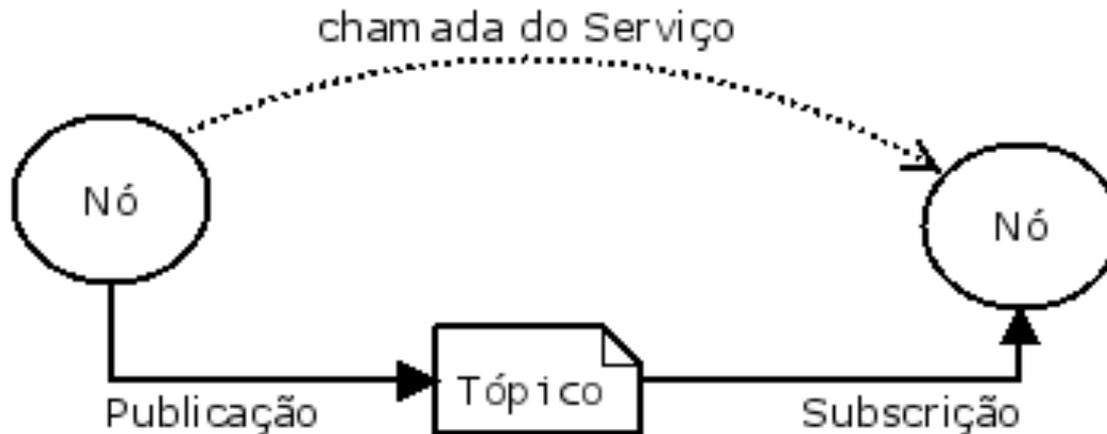
```
$ roslaunch turtlesim turtlesim_node
```

Serviços

- O serviço de um determinado nó pode ser iniciado por

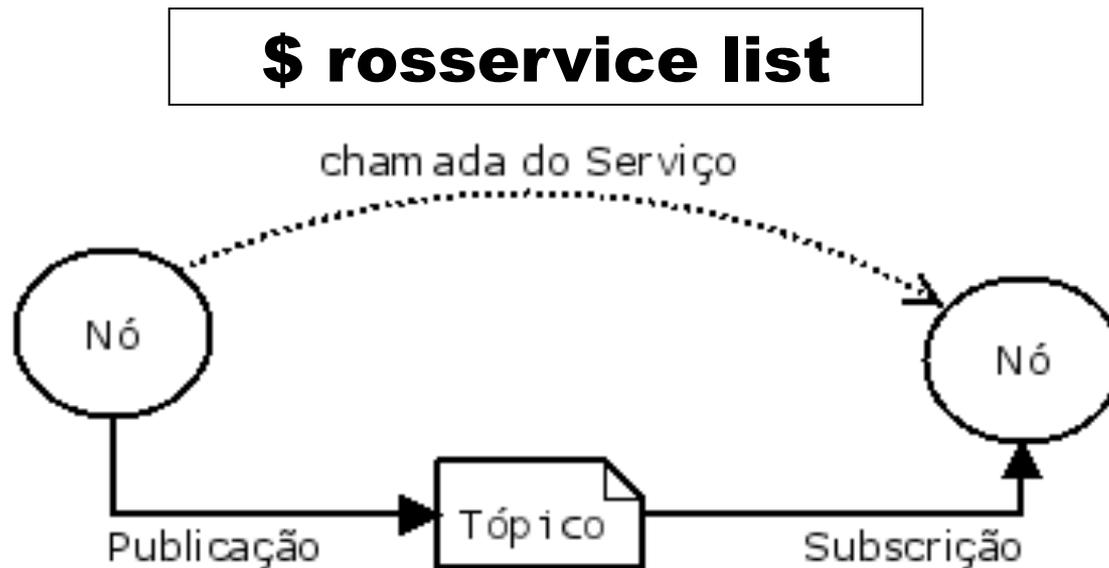
\$ rosservice call [nome_do_serviço]

\$ rosservice call /reset



Serviços

- São uma forma de interação direta entre dois ou mais nós
- A lista de serviços disponíveis pode ser acessado por



TurtleSim

Ver os tópicos ativos

```
$ rostopic list
```

Topicos do TurtleSim

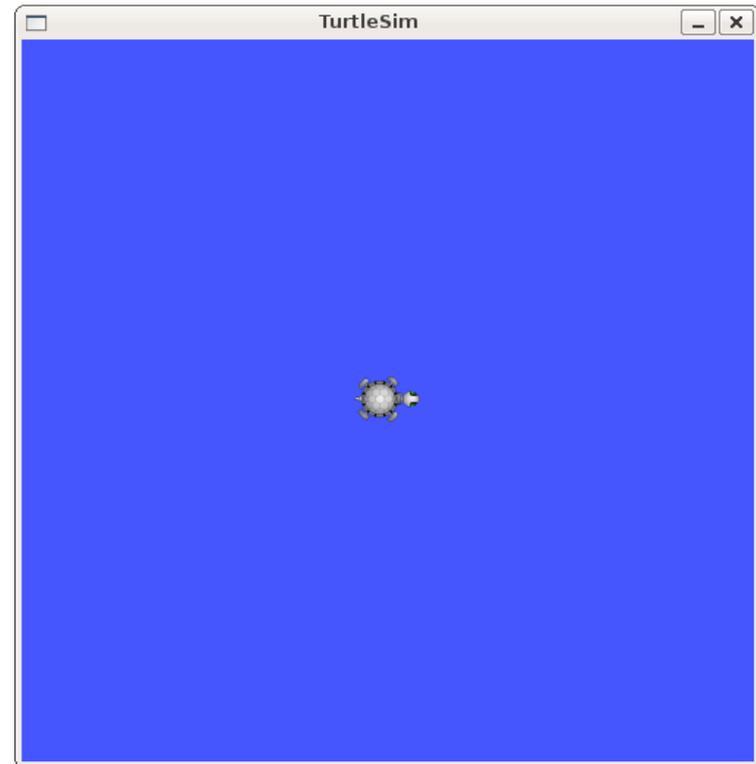
```
/rosout
```

```
/rosout_agg
```

```
/turtle1/cmd_vel
```

```
/turtle1/color_sensor
```

```
/turtle1/pose
```



Criar um pacote

- A criação de pacotes é realizada dentro do workspace ROS na pasta src (source)

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

- Para criar um pacote

```
$ catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
```

- Criando o pacote do tutorial

```
$ catkin_create_pkg tutorial_ros roscpp geometry_msgs turtlesim
```

Compilação do Workspace ROS

- É realizada na pasta principal do workspace

```
$ cd ~/catkin_ws
```

- Através do comando

```
$ catkin_make
```

Publicadores (Publishers)

- O publisher é um nó ROS que escreve mensagens em um determinado tópico
- A criação de novos nós realizada na pasta src (source) de um pacote

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

- Pode ser utilizado qualquer editor de texto

```
$ nano 1.publisher.cpp
```

Publicadores (Publishers)

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

int main(int argc, char **argv) {
    ros::init(argc, argv, "exemplo_publisher");
    ros::NodeHandle n;
    ros::Publisher 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 << "mensagem " << count;
        msg.data = ss.str();
        ROS_INFO("[Enviado] %s", msg.data.c_str());
        pub.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
        ++count;
    }
    return 0;
}
```

Compilar o novo nó

- É necessário adicionar ao arquivo CMakeLists.txt

```
$ cd ~/catkin_ws/tutorial_ros  
$ nano CMakeLists.txt
```

- Deve-se adicionar as seguintes linhas no final do arquivo:

```
add_executable(1.publicar src/1.publisher.cpp)  
target_link_libraries(1.publicar ${catkin_LIBRARIES})
```

Compilar o novo nó

- Compilar o Workspace novamente

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

- O novo nó pode ser executado

```
$ rosrun tutorial_ros 1.publicar
```

Subscrição (Subscriber)

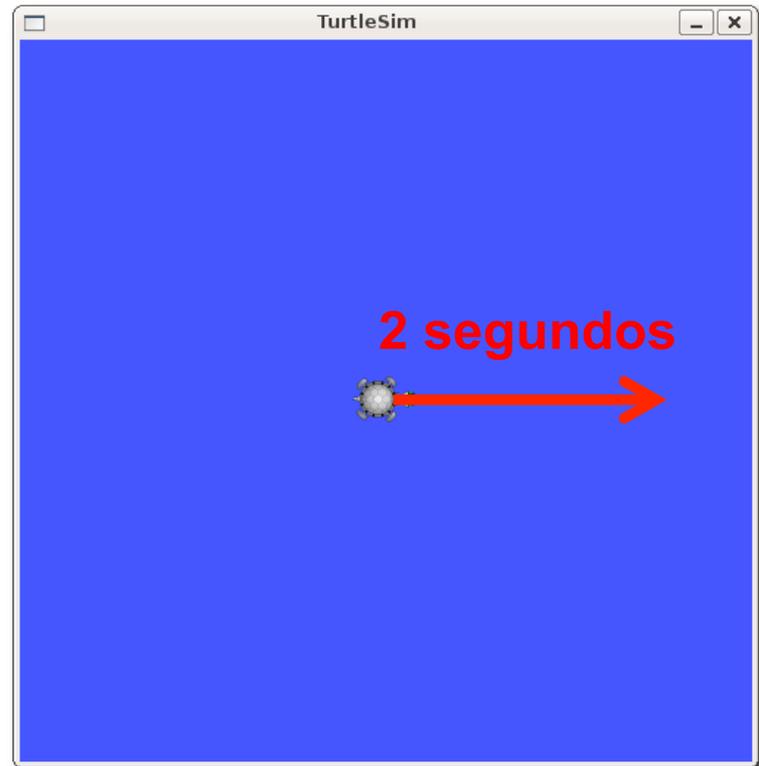
```
#include "ros/ros.h"
#include "std_msgs/String.h"

void subCallback(const std_msgs::String::ConstPtr& msg) {
    ROS_INFO("Recebido: [%s]", msg->data.c_str());
}

int main(int argc, char **argv) {
    ros::init(argc, argv, "exemplo_subscriber");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("chatter", 1000, subCallback);
    ros::spin();
    return 0;
}
```

Exemplo 1: Movimento simples

- **Objetivo:** realizar o movimento linear para frente por 2 segundos e parar na sequencia



Exemplo 1: Movimento simples

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "turtlesim/Pose.h"
#include <iostream>

using namespace std;
int main(int argc, char **argv)
{
    ros::init(argc, argv, "turtle_movimento_simples");
    ros::NodeHandle n;
    ros::Publisher pub = n.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    ros::Rate loop_rate(10);
```

Exemplo 1: Movimento simples

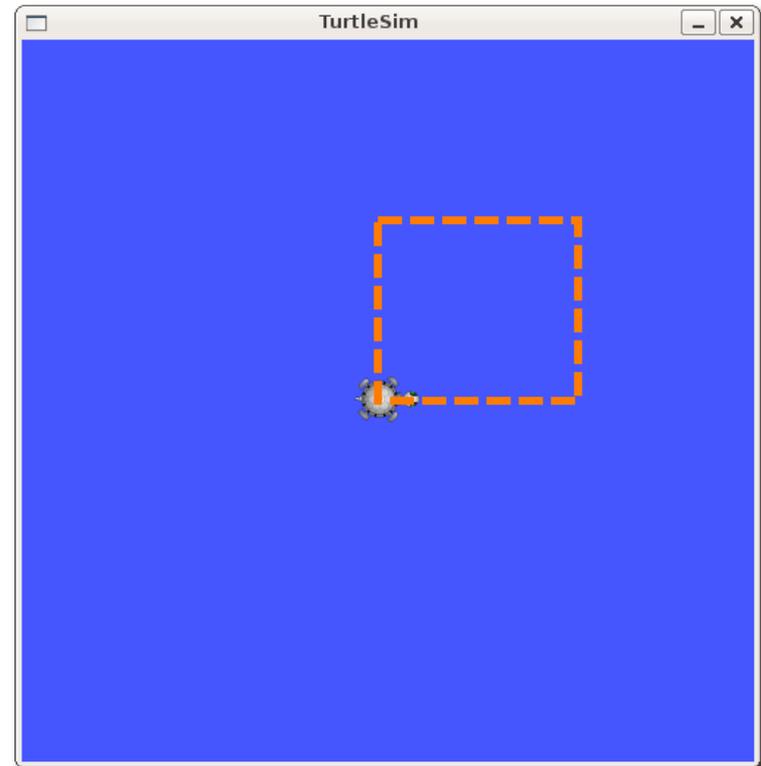
```
if (ros::ok())
{
    geometry_msgs::Twist msg;
    int l;
    ROS_INFO("Pressione qualquer tecla para iniciar...");
    system("rosservice call reset");
    getchar();
    ros::spinOnce();

    msg.linear.x = 2;
    pub.publish(msg);
    ros::spinOnce();
    sleep(2);

    msg.linear.x = 0;
    pub.publish(msg);
    ros::spinOnce();
    ROS_WARN("Trajeto finalizado...");
}
return 0;
}
```

Exemplo 2: Movimento quadrado

- **Objetivo:** realizar uma sequencia de movimentos com o robô turtle para execução de uma trajetória quadrada



Exemplo 2: Movimento quadrado

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "turtlesim/Pose.h"
#include <iostream>
#include <math.h>
using namespace std;

int main(int argc, char **argv) {
    ros::init(argc, argv, "turtle_controle_velocidade");
    ros::NodeHandle n;
    ros::Publisher pub = n.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
    ros::Rate loop_rate(10);
    if (ros::ok()) {
        geometry_msgs::Twist msg;
        int i;
        ros::spinOnce();
        ROS_INFO("Pressione qualquer tecla para iniciar...");
        system("rosservice call reset");
        getchar();
        ros::spinOnce();
    }
}
```

Exemplo 2: Movimento quadrado

```
for(i=0;i<4;i++){
    msg.linear.x = 2;
    msg.angular.z = 0;
    pub.publish(msg);
    ros::spinOnce();
    sleep(1);

    msg.linear.x = 0;
    msg.angular.z = M_PI/2;
    pub.publish(msg);
    ros::spinOnce();
    sleep(1);
}
ROS_WARN("Quadrado finalizado...");
}
return 0;
}
```

Referências

- Instalação do ROS Indigo
<http://wiki.ros.org/indigo/Installation/Ubuntu>
- Tutoriais do ROS [Sugestão do 1 até 12]
[http://wiki.ros.org/pt_BR/ROS/Tutorials/
InstallingandConfiguringROSEnvironment](http://wiki.ros.org/pt_BR/ROS/Tutorials/InstallingandConfiguringROSEnvironment)