Introdução à Robótica

através do Robot Operating System (ROS)

Prof. André Schneider de Oliveira

Universidade Técnológica Federal do Paraná (UTFPR)

Robótica









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

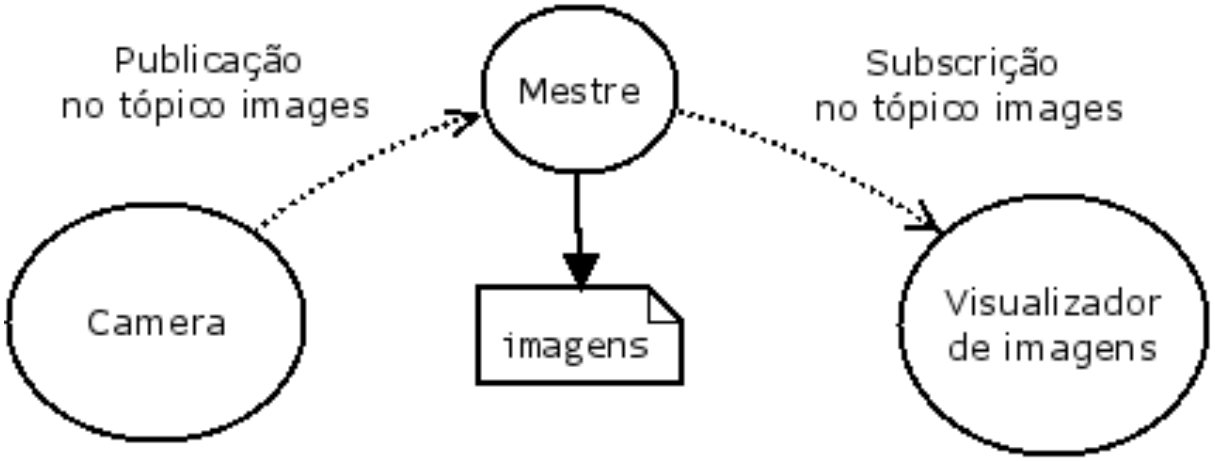
Robot Operating System (ROS)

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

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

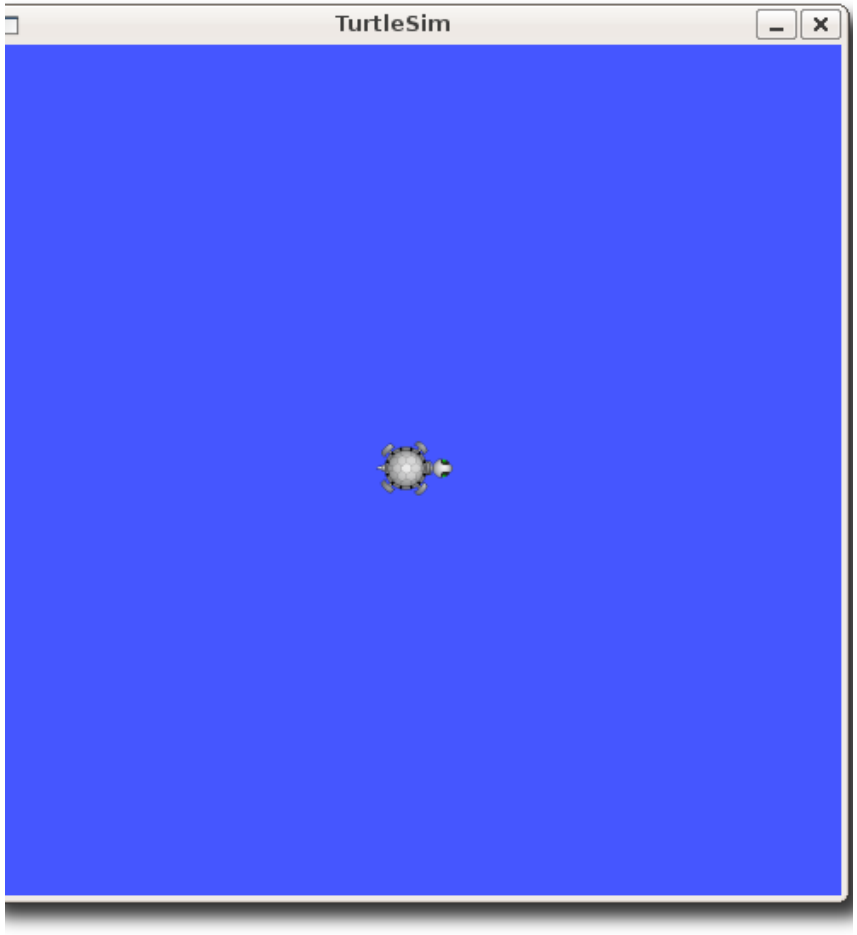
$ roscore

```
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
 * /rosdistro: 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

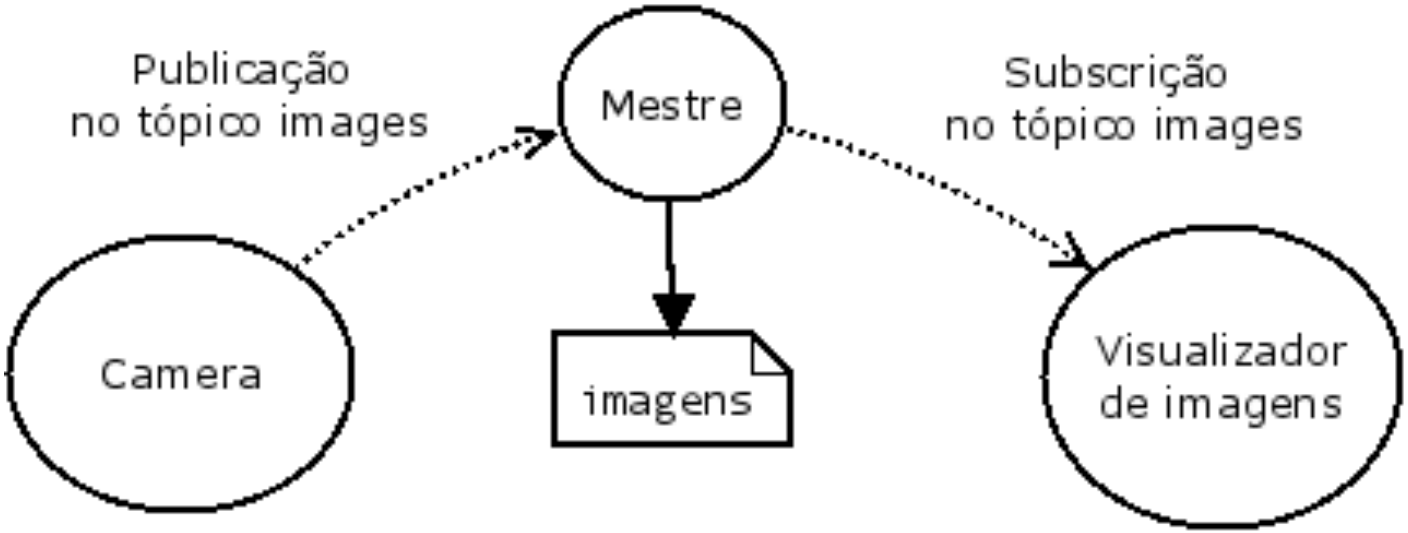
- Simulador integrado do ROS
- Executado pelo comando

$ *rosrun 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

```
$ rosmsg list
```

- Ver o conteúdo de uma mensagem

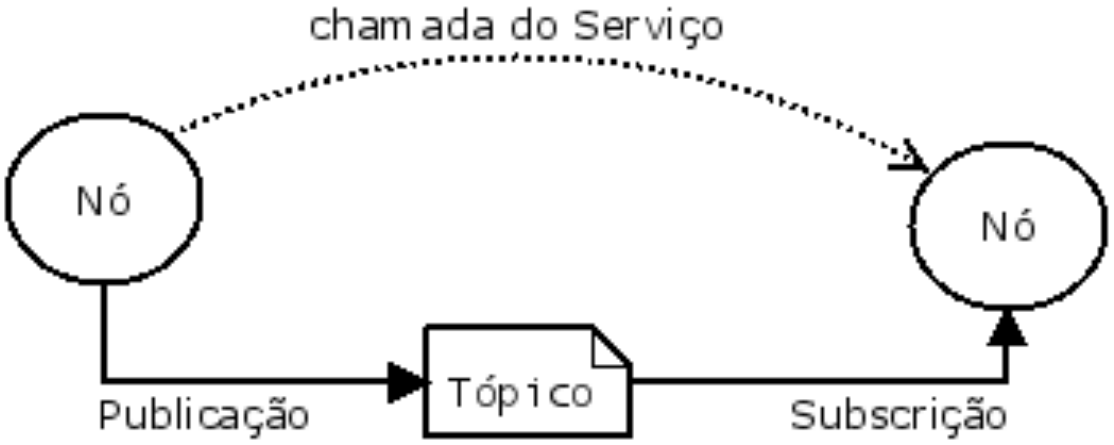
```
$ rosmsg show [tipo_da_msg]
```

```
$ rosmsg 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 visualizador 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

```
$ rosnode list
```

- Um nó é executado por

```
$ rosrun [nome_do_pacote] [nome_do_nó]
```

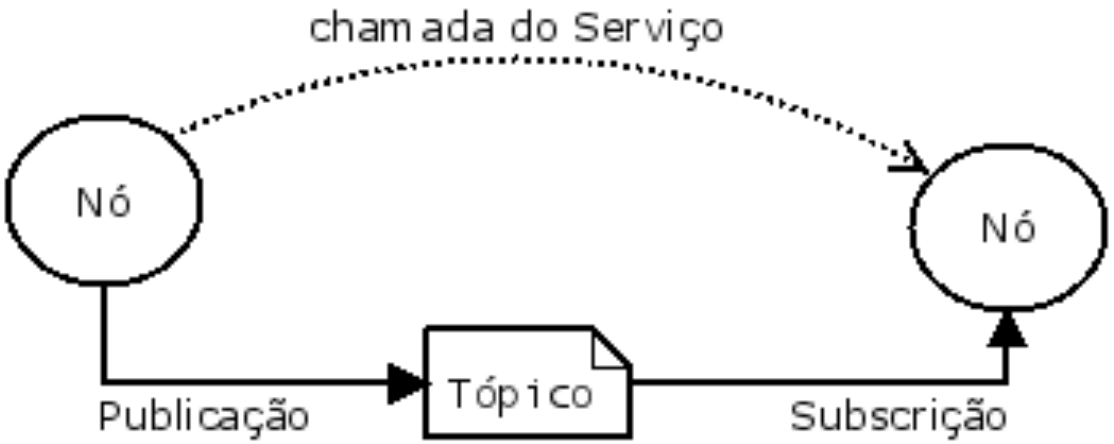
```
$ rosrun turtlesim turtlesim_node
```

Serviços

- O serviço de um determinado nó pode ser iniciado por

$ rosservice call [nome_do_serviço]

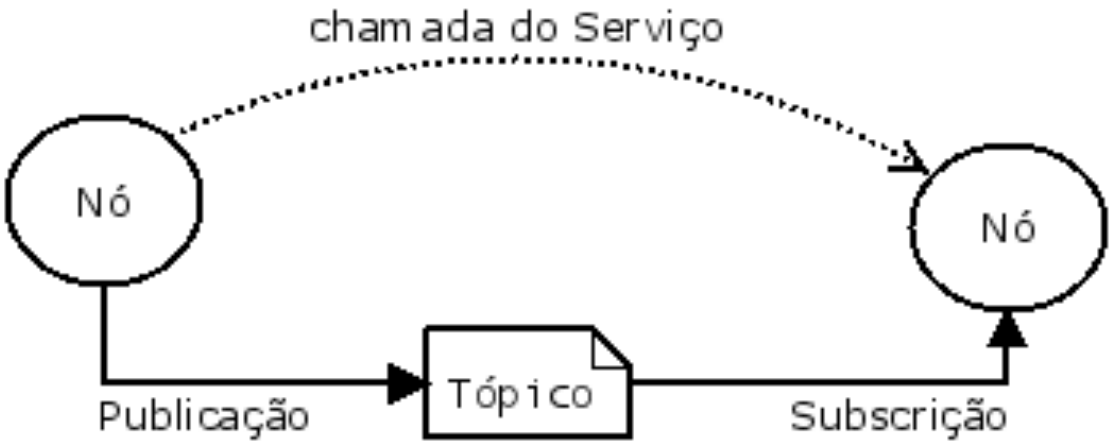
$ rosservice call /reset



Serviços

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

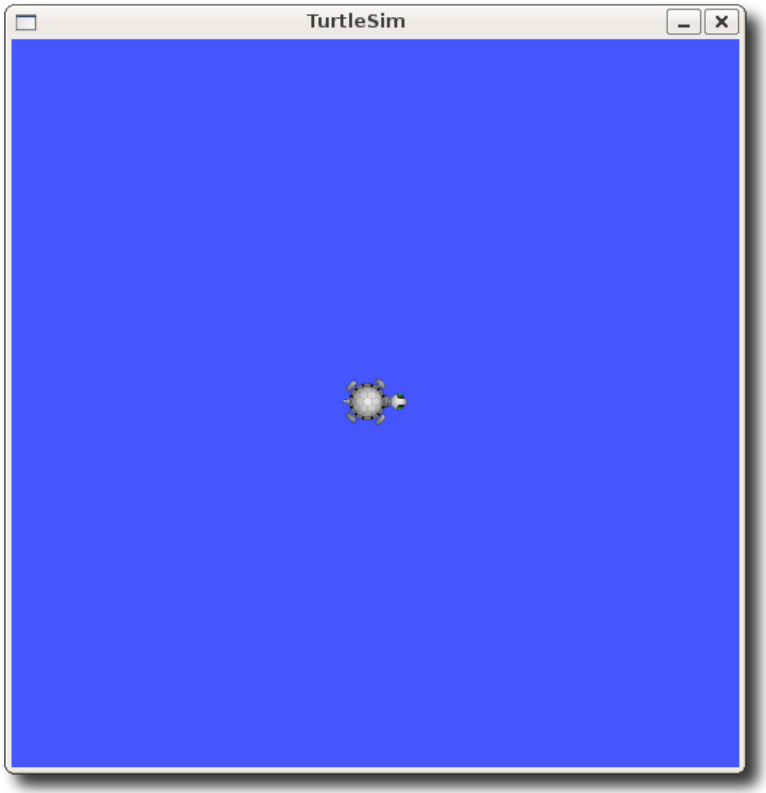
$ rosservice list



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;
}
```

https://raw.githubusercontent.com/RoboticaAI/tutorial_ros/master/src/1.publisher.cpp

Compilar o novo nó

- É necessário adicionar ao arquivo CMakeLists.txt

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

- 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
```

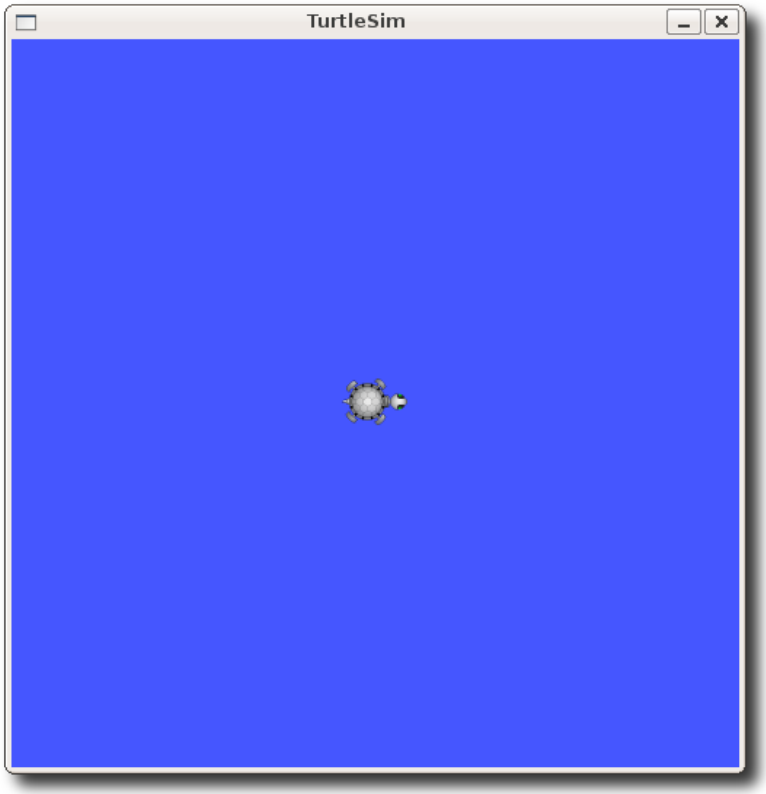
Subscrição (Subscriber)

```
#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;
}
```

https://raw.githubusercontent.com/RoboticaAI/tutorial_ros/master/src/2.subscriber.cpp

Teleoperação pelo teclado

- **Objetivo:** controlar o robô turtle através das teclas [a,s,d,w] do teclado



Teleoperação pelo teclado

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

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

        geometry_msgs::Twist msg;
        char tecla;
```

https://raw.githubusercontent.com/RoboticaAI/tutorial_ros/master/src/kbhit.h

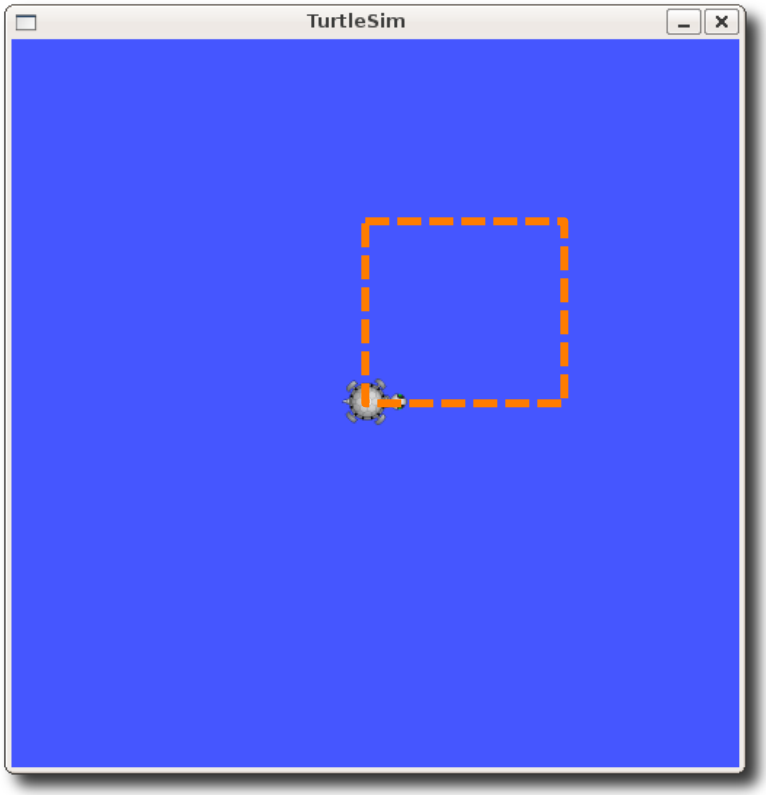
Teleoperação pelo teclado

```
while (ros::ok() && tecla != 'q') {
        if (kbhit())
                tecla = getchar();
        if (tecla == 'w'){ msg.linear.x = 0.5; msg.angular.z = 0; ROS_INFO("Frente"); }
        if (tecla == 's'){ msg.linear.x = -0.5; msg.angular.z = 0; ROS_INFO("Traz"); }
        if (tecla == 'a'){ msg.angular.z = 0.5; ROS_INFO("Esquerda"); }
        if (tecla == 'd'){ msg.angular.z = -0.5; ROS_INFO("Direita"); }
        if (tecla == 'q'){ msg.linear.x = 0; msg.angular.z = 0; ROS_INFO("Parado"); }

         pub.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
        }
return 0;
}
```

https://raw.githubusercontent.com/RoboticaAI/tutorial_ros/master/src/3.teleop.cpp

Sequência de movimentos

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



Sequência de movimentos

```
#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();
```

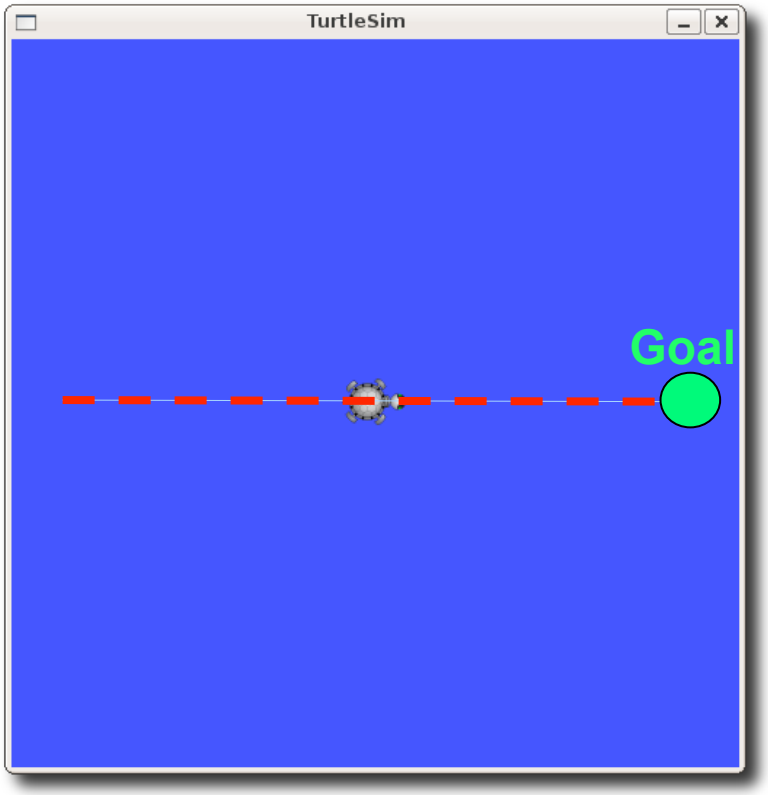
Sequência de movimentos

```
                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;
}
```

https://raw.githubusercontent.com/RoboticaAI/tutorial_ros/master/src/4.ctrl_velocidade.cpp

Controle de posição em X

- **Objetivo:** realizar o controle do movimento do robô turtle ao longo do eixo X para atingir uma determinada posição (Goal)



Controle de posição em X

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

turtlesim::Pose feedback;

void subCallback(const turtlesim::Pose::ConstPtr& msg) {
        feedback.x = msg->x;
        feedback.y = msg->y;
        feedback.theta = msg->theta;
        feedback.linear_velocity = msg->linear_velocity;
        feedback.angular_velocity = msg->angular_velocity;
}
```

Controle de posição em X

```
int main(int argc, char **argv) {
        ros::init(argc, argv, "turtle_controle_X");
        ros::NodeHandle n;
        ros::Publisher pub = n.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
        ros::Subscriber sub = n.subscribe("turtle1/pose", 1000, subCallback);
        ros::Rate loop_rate(10);
        system("rosservice call reset");

        if (ros::ok()) {
                geometry_msgs::Twist msg;
                float desejado, erro=99;
                float tolerance = 0.01;
                float Kpos = 10; cout << "Digite a posicao\nX>>";
                cin >> desejado;
```

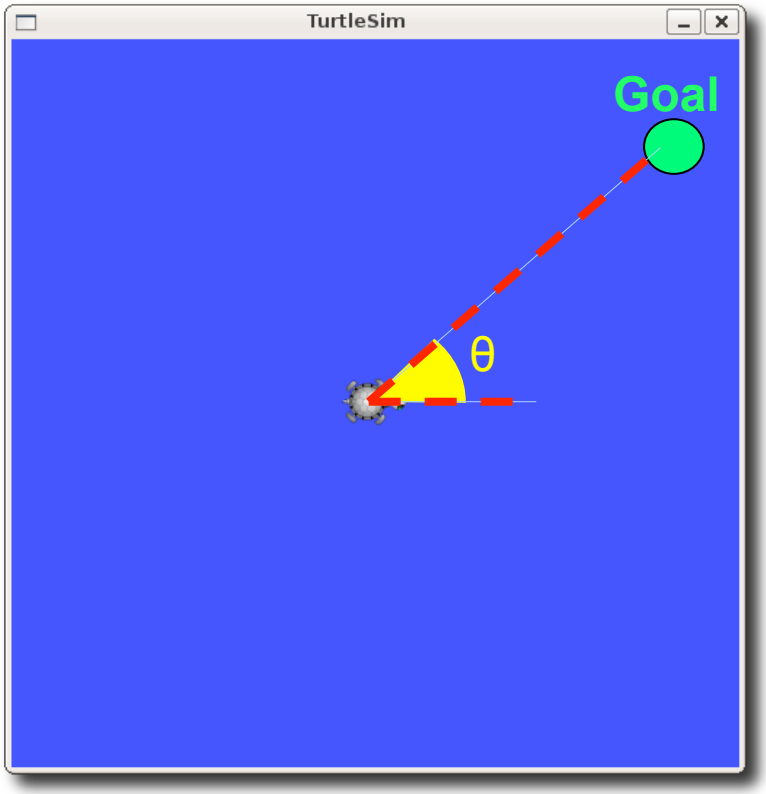
Controle de posição em X

```
                    while (abs(erro) > tolerance) {
                            erro = desejado-feedback.x;
                            msg.linear.x = Kpos*erro/(60/10);
                            ROS_INFO("X>>%f,Erro>>%f",feedback.x,erro);
                            pub.publish(msg);
                            ros::spinOnce();
                            loop_rate.sleep();
                    }
                    ROS_WARN("...Posicao alcancada...");
            }
return 0;
}
```

https://raw.githubusercontent.com/RoboticaAI/tutorial_ros/master/src/5.ctrl_X.cpp

Controle de posição em X e Y

- **Objetivo:** realizar o controle do movimento do robô turtle ao longo dos eixos X e Y para atingir uma determinada posição (Goal)



Controle de posição em X e Y

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

turtlesim::Pose feedback;

void subCallback(const turtlesim::Pose::ConstPtr& msg) {
        feedback.x = msg->x;
        feedback.y = msg->y;
        feedback.theta = msg->theta;
}
```

Controle de posição em X e Y

```
//Controle de posicao
            while (dist > tolerance_pos){
                        dist = sqrt(pow(posdesejada[0]-feedback.x,2)+pow(posdesejada[1]-feedback.y,2));
                         msg.linear.x = abs(Kpos*(dist) /(60/10));
                         ROS_INFO("Dist>>%f",dist);
                        pub.publish(msg);
                        ros::spinOnce();
                        loop_rate.sleep();
            }
            ROS_WARN("...Posicao alcancada...");
      }
return 0;
}
```

https://raw.githubusercontent.com/RoboticaAI/tutorial_ros/master/src/6.ctrl_posicao.cpp

Controle de posição em X e Y

```
// Controle da orientacao
        while (abs(erroorie) > tolerance_orie) {
                erroorie = angulo-feedback.theta;
                msg.angular.z = Korie*erroorie/(60/10);
                ROS_INFO("theta>>%f,Erro>>%f",feedback.theta,erroorie);
                pub.publish(msg);
                ros::spinOnce();
                loop_rate.sleep();
        }
        msg.angular.z = 0;
        pub.publish(msg);
        ros::spinOnce();
        ROS_WARN("...Orientacao alcancada...");
```

Controle de posição em X e Y

```
int main(int argc, char **argv) {
        ros::init(argc, argv, "turtle_controle_posicao");
        ros::NodeHandle n;
        ros::Publisher pub = n.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1000);
        ros::Subscriber sub = n.subscribe("turtle1/pose", 1000, subCallback);
        ros::Rate loop_rate(10);

        if (ros::ok()) {
                geometry_msgs::Twist msg;
                float posdesejada[2], oridesejada, dist=99, erroorie=99;
                float tolerance_orie = 0.005, tolerance_pos = 0.05;
                float Kpos = 10, Korie = 15;
                float angulo;

                cout << "Digite a posicao\nX>>";
                cin >> posdesejada[0];
                cout << "Y>>"; cin >> posdesejada[1];
                ros::spinOnce();
                angulo = atan2(posdesejada[1]-feedback.y,posdesejada[0]-feedback.x);
                ROS_WARN("angulo>>%f\n",angulo);
```

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