

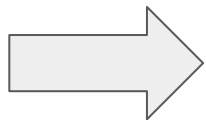


RVIZ

Ferramenta de Visualização

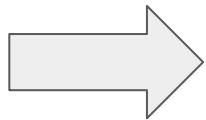
Kristofer Stift Kappel
kskappel@inf.ufpel.edu.br

Repositório



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

Tutorial baseado nos links:



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

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

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

Prelúdio

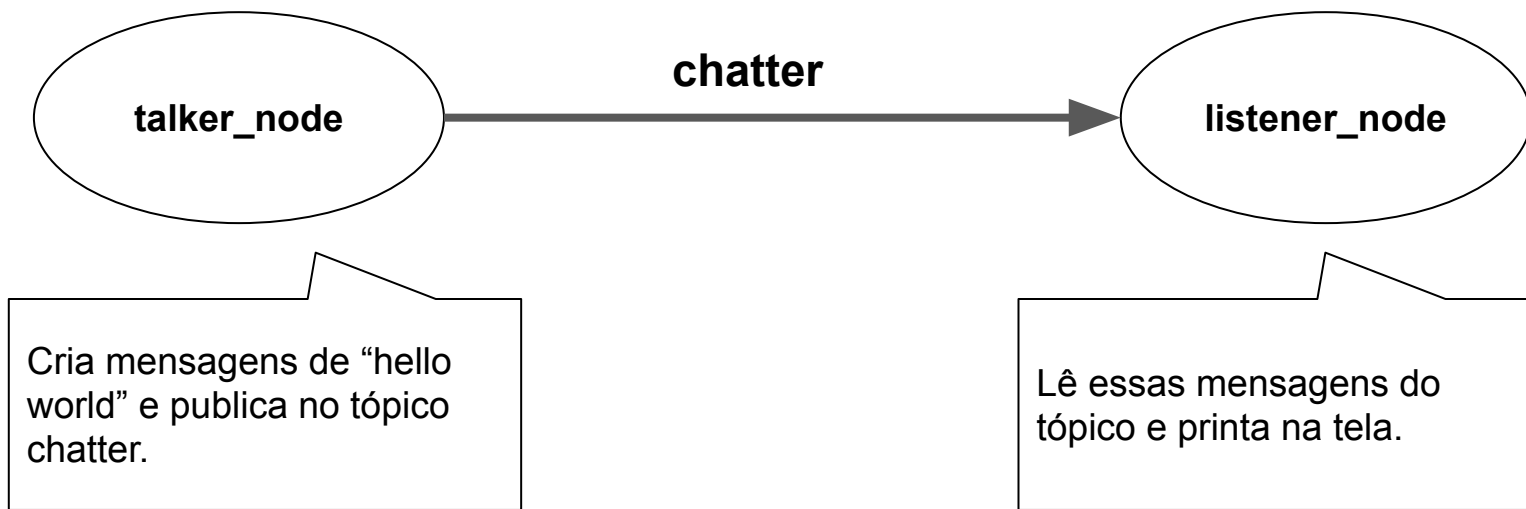
- Continuação do slide de Introdução ao ROS. Caso necessário use o link do slide anterior para acessar o slide de Introdução.
 - Caso tenha seguido os passos do slide de Introdução você deve ter um workspace já criado com um pacote chamado “minicurso”.
 - Caso não tenha seguido o slide citado você deve criar um workspace e dentro do src criar um pacote chamado minicurso.

Prelúdio

- O primeiro passo é aprender a lidar com o ROS utilizando linguagens de programação.
- Os próximos slides apresentam exemplos simples em C++ utilizando ros topics.
- Um dos códigos representa um **Publisher**, que vai publicar mensagens em um tópico.
- O outro representa um **Subscriber**, que vai ler essas mensagens e printar na tela.

Prelúdio

- Os códigos a seguir (talker.cpp e listener.cpp) compõe 2 **nodos** que vão se comunicar da seguinte maneira:



Código Publisher C++ - talker.cpp

```
#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 Subscriber C++ - listener.cpp

```
#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 poder 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:

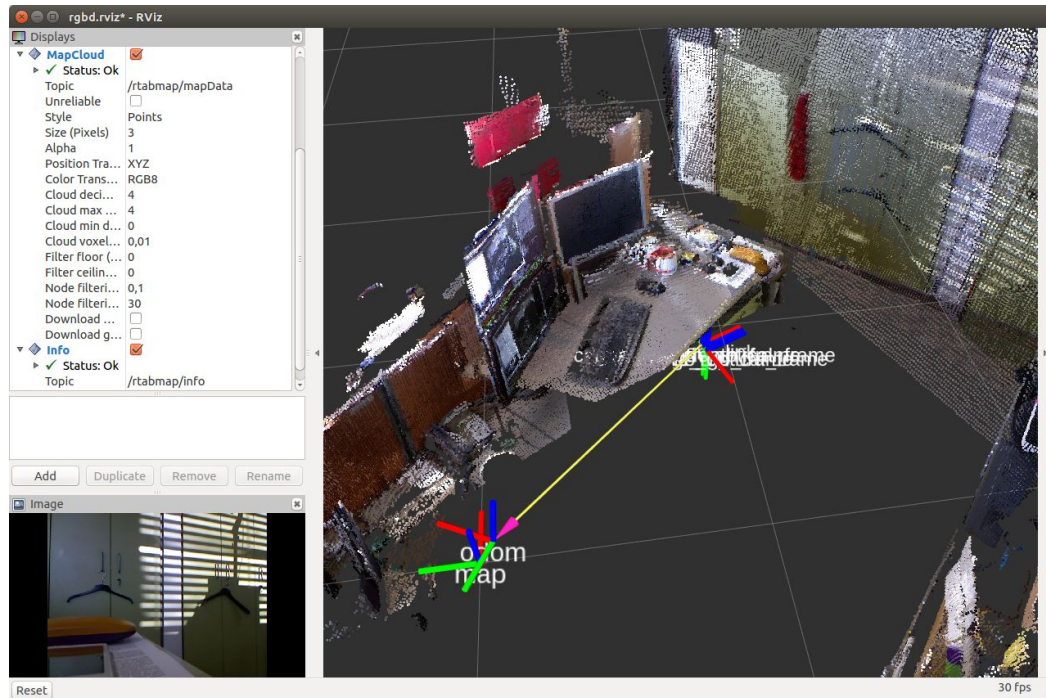
```
$ rosrunc minicurso talker_node
```

Terminal 3:

```
$ rosrunc minicurso listener_node
```

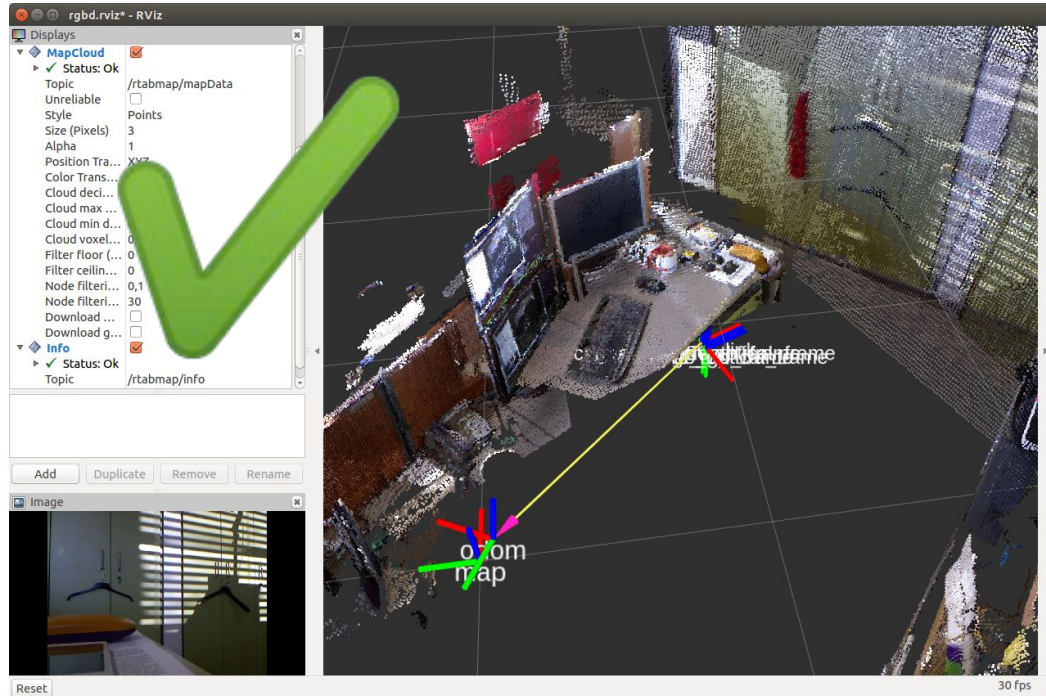
RVIZ?

```
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
```



RVIZ?

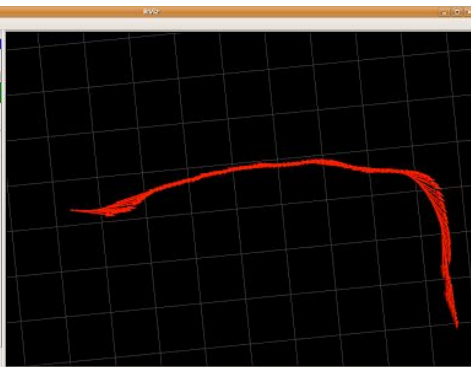
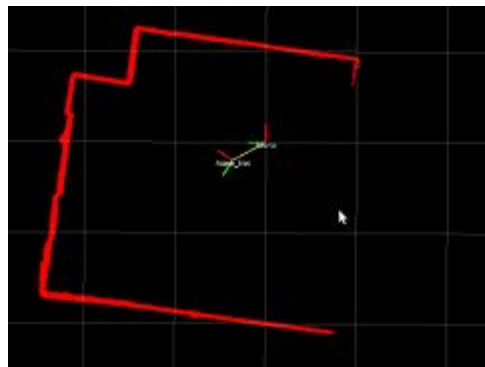
```
std_msgs/Header header
uint32 seq
time stamp
string frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/Pose pose
geometry_msgs/Point position
float64 x
float64 y
float64 z
geometry_msgs/Quaternion orientation
float64 x
float64 y
float64 z
float64 w
float64[36] covariance
geometry_msgs/TwistWithCovariance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



RVIZ!

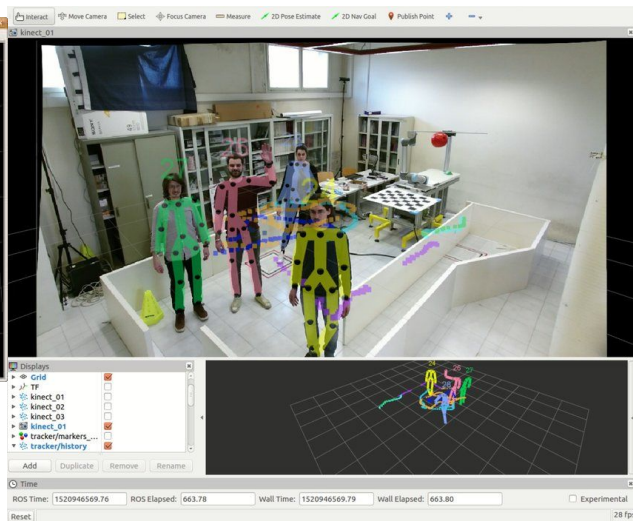
- Dados no RViZ podem ser de vários tipos.

LaserScan



Odometry

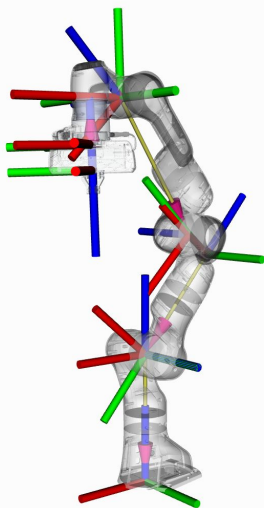
Image



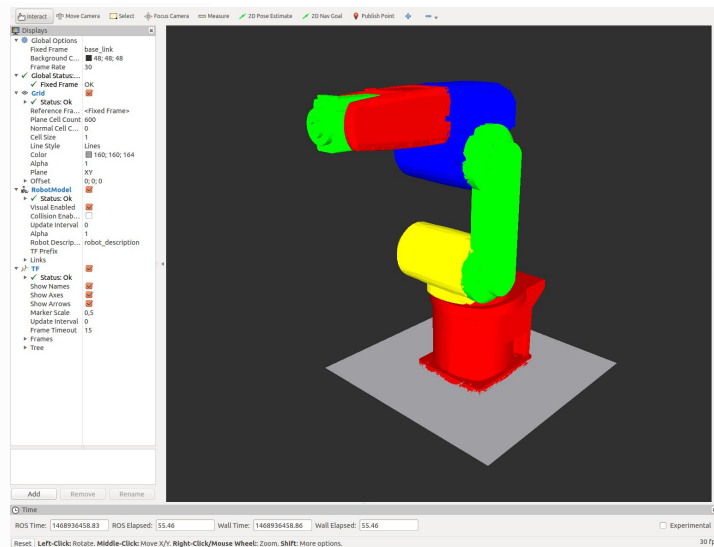
RVIZ!

- Dados no RViZ podem ser de vários tipos.

Axis



Joints



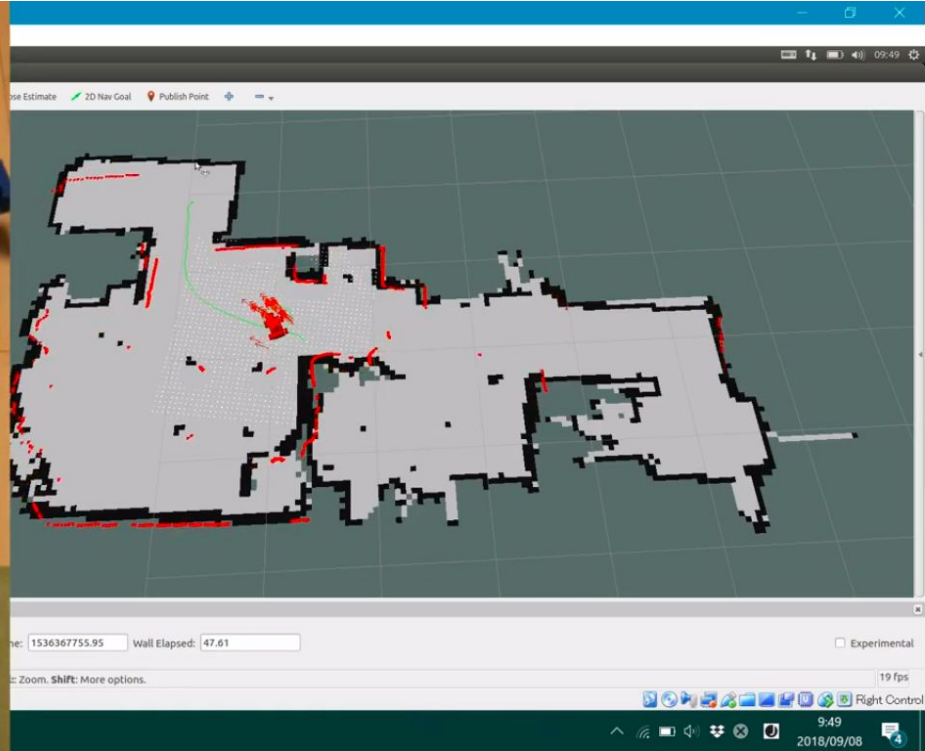
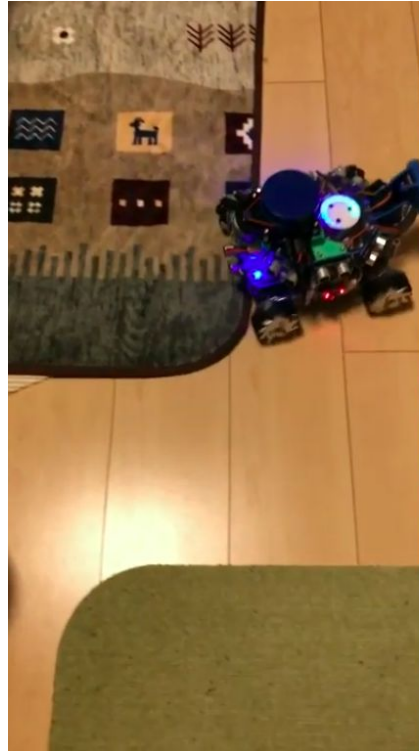
Point Clouds



RVIZ!

- Navigation Stack:

<https://www.youtube.com/watch?v=Gsx4gJVk0UY>

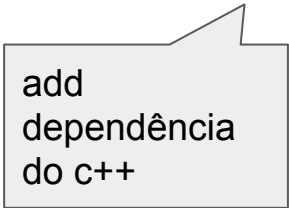


Markers

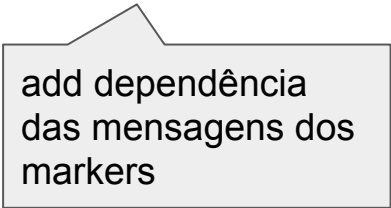
Vamos fazer um código que mostre markers na tela:

Começaremos criando um pacote chamado “ex_markers” dentro da pasta src do ws.

```
$ catkin_create_pkg ex_markers roscpp visualization_msgs
```



add
dependência
do c++



add dependência
das mensagens dos
markers

Markers

Dentro do src do pacote, crie um arquivo cpp chamado “basic_shapes.cpp”. Iremos começar pelo básico do código.

```
#include "ros/ros.h"  
#include <visualization_msgs/Marker.h>
```

```
int main(int argc, char **argv)  
{  
    ros::init(argc, argv, "basic_shapes");  
    ros::NodeHandle n;  
    ros::Rate r(10);  
}
```

Markers

Adicionando um publisher pros markers

```
#include "ros/ros.h"
#include <visualization_msgs/Marker.h>

int main(int argc, char **argv)
{
    ros::init(argc, argv, "basic_shapes");
    ros::NodeHandle n;
    ros::Rate r(10);
    ros::Publisher marker_pub = n.advertise<visualization_msgs::Marker>("visualization_marker", 1);
    ....
}
```

Markers

Agora precisamos ver o que compõe uma mensagem de marker antes de publicar.

```
$ rosmmsg show visualization_msgs/Marker
```

Para criar uma mensagem de marker para ser publicada cada campo referente ao tipo deve ser preenchido.

Markers

```
ros::Publisher marker_pub = n.advertise<visualization_msgs::Marker>("visualization_marker", 1);
```

```
visualization_msgs::Marker marker;  
marker.header.frame_id = "/my_frame";  
marker.header.stamp = ros::Time::now();  
marker.ns = "basic_shape";  
marker.id = 0;  
marker.type = visualization_msgs::Marker::CUBE;  
marker.action = visualization_msgs::Marker::ADD;  
marker.pose.position.x = 0;  
marker.pose.position.y = 0;  
marker.pose.position.z = 0;  
marker.pose.orientation.x = 0.0;  
marker.pose.orientation.y = 0.0;  
marker.pose.orientation.z = 0.0;  
marker.pose.orientation.w = 1.0;
```

```
.  
.  
.
```

Markers

```
.  
.   
.   
marker.scale.x = 1.0;  
marker.scale.y = 1.0;  
marker.scale.z = 1.0;  
marker.color.r = 0.0f;  
marker.color.g = 1.0f;  
marker.color.b = 0.0f;  
  marker.color.a = 1.0;  
marker.lifetime = ros::Duration();  
marker_pub.publish(marker);  
r.sleep();
```

Agora, de mesmo modo, adicione no CMakeLists.txt do pacote ex_markers. (igual ao slide 8, porém troque o nome do arquivo por basic_shapes)

Depois compile novamente com o catkin_make. (cd ~/catkin_ws & catkin_make)

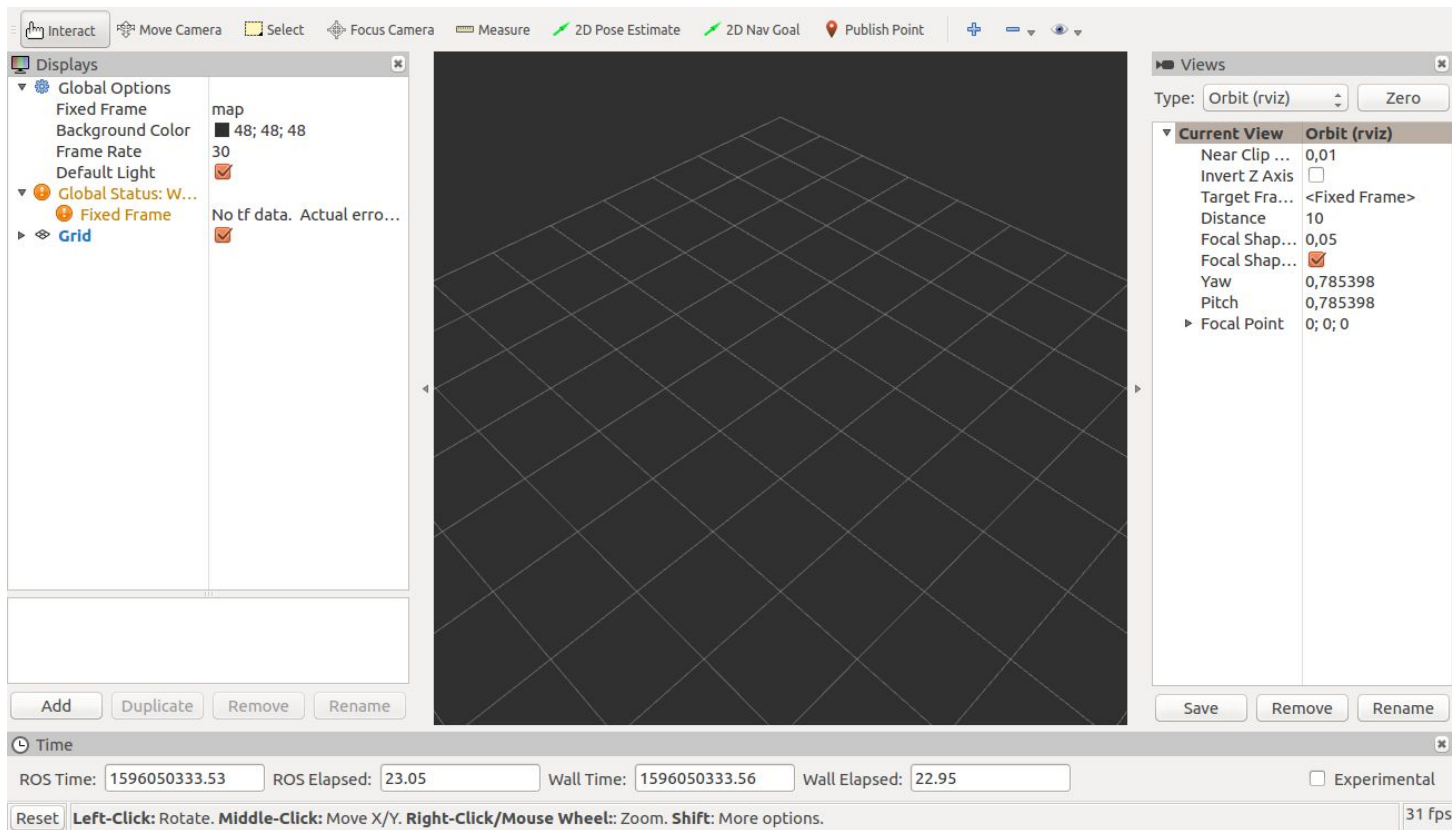
Markers

Para rodar o rviz:

\$ rosrn rviz rviz

E você deve de estar

vendo uma tela assim:

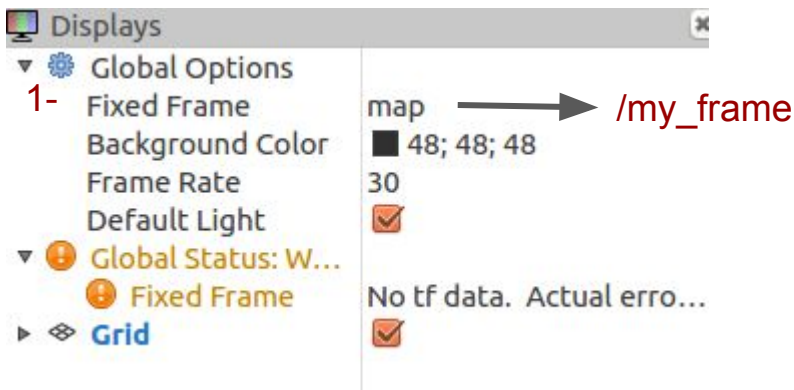


Markers

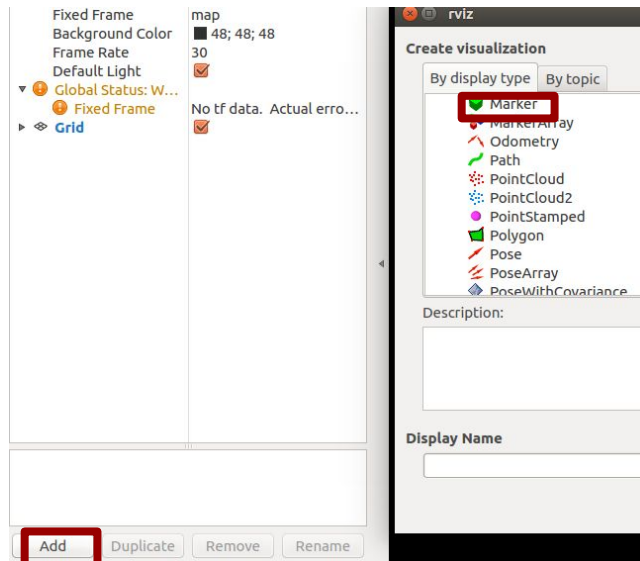
Para poder ver os markers é necessário mudar poucas coisas no RVIZ

1- Uma recomendação é mudar o fixed frame para /my_frame, que é o frame indicado no código do marker.

2- Para que o marker seja exibido é necessário adicionar um visualizador de markers no botão Add no canto inferior esquerdo e depois selecionar Marker.



2-



Markers

Agora para rodar o código do marker:

```
$ rosrun ex_markers basic_shape
```

E o que acontece?

Markers

Agora para rodar o código do marker:

```
$ rosrun ex_markers basic_shape
```

E o que acontece? **NADA**

Markers

Agora para rodar o código do marker:

```
$ roslaunch ex_markers basic_shape
```

E o que acontece? **NADA**

Isso se dá pela natureza paralela da ROS, em que existe atraso na comunicação com o master, tanto para criar o tópico como para mandar a mensagem e o processo terminar antes do fim das ações.

Markers

Para corrigir esse problema, podemos, por exemplo, adicionar esse trecho de código antes de publicar:

```
while (marker_pub.getNumSubscribers() < 1)  
  {  
    if (!ros::ok())  
    {  
      return 0;  
    }  
    ROS_WARN_ONCE("esperando um subscriber");  
    sleep(1);  
  }
```

Esse trecho espera até ter um subscriber conectado ao tópico antes de prosseguir.

Markers

Para visualizar diferentes markers podemos trocar a forma do cubo. Para isso são necessárias essas modificações:

```
ros::Publisher marker_pub = n.advertise<visualization_msgs::Marker>("visualization_marker", 1);
```

```
uint32_t shape = visualization_msgs::Marker::CUBE;  
while (ros::ok())  
{
```

Trocar **marker.type = visualization_msgs::Marker::CUBE;** por **marker.type = shape;**

Não esqueça de adicionar um **}** no final do arquivo

Markers

Para visualizar diferentes markers podemos trocar a forma do cubo. Para isso são necessárias essas modificações:

```
marker_pub.publish(marker);
```

```
switch (shape)
```

```
{  
  case visualization_msgs::Marker::CUBE:  
    shape = visualization_msgs::Marker::SPHERE;  
    break;  
  case visualization_msgs::Marker::SPHERE:  
    shape = visualization_msgs::Marker::ARROW;  
    break;  
  case visualization_msgs::Marker::ARROW:  
    shape = visualization_msgs::Marker::CYLINDER;  
    break;  
  case visualization_msgs::Marker::CYLINDER:  
    shape = visualization_msgs::Marker::CUBE;  
    break;  
}
```

Agora compile com `catkin_make` e execute!



RVIZ

Ferramenta de Visualização

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