

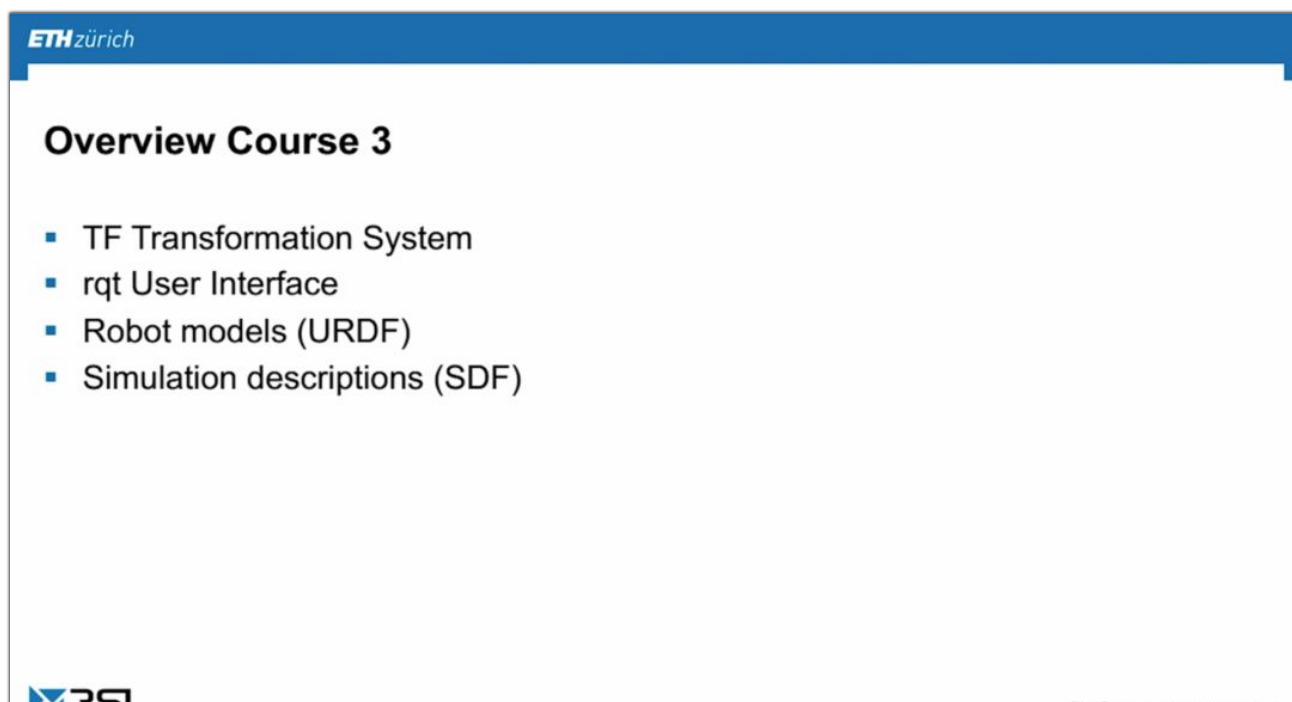
进阶课程^{③④} | Apollo ROS原理—3

机器人操作系统(ROS)是一个成熟而灵活的机器人编程框架。ROS提供了所需的工具，可以轻松访问**传感器数据**，**处理数据**，并为机器人的电机和其它执行器生成适当的响应。整个ROS系统被设计为在计算方面完全分布，因此不同的计算机可以参与控制过程，并作为单个实体(机器人)一起行动。

目前ROS仅适用于Apollo 3.0之前的版本，最新代码及功能还请参照Apollo 3.5及5.0版本。

以下，ENJOY

本节主要讲解四个比较基础的方面，如图所示。第一是TF坐标系转换，它在自动驾驶中用的非常广泛；第二是RQT用户接口；第三是机器人模型；第四是仿真描述。



ETH zürich

Overview Course 3

- TF Transformation System
- rqt User Interface
- Robot models (URDF)
- Simulation descriptions (SDF)

RSI

Stefan Edelkamp - 1. September 2016 - 1 / 4

TF坐标系转换

为什么需要TF坐标系转换？因为自动驾驶使用的ROS架构是一个松耦合关系，每个节点独立运行，节点有一套自己的XYZ坐标系，当把他们组装到一块时，每个节点的坐标系都是相对独立的，但整个自动驾驶系统需要把每一个节点所使用的信息和一些参数转化到同一个世界坐标系里。TF节点就提供了对应的坐标系转换功能，TF消息也是通过基于Message的订阅和发布消息来完成的。

ETH zürich

TF Transformation System

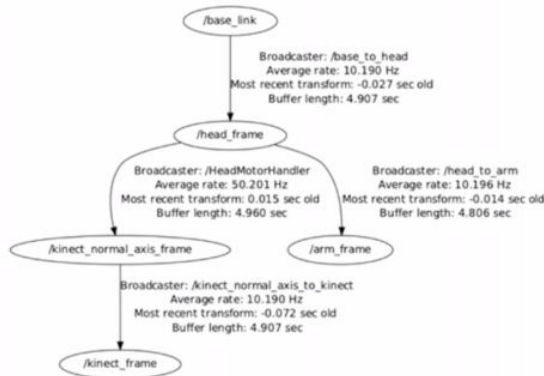
Transform Tree

- TF listeners use a buffer to listen to all broadcasted transforms
- Query for specific transforms from the transform tree

tf2_msgs/TFMessage.msg

```

geometry_msgs/TransformStamped[] transforms
std_msgs/Header header
uint32 seqtime stamp
string frame_id
string child_frame_id
geometry_msgs/Transform transform
geometry_msgs/Vector3 translation
geometry_msgs/Quaternion rotation
    
```



```

graph TD
    base_link["/base_link"] -- "Broadcaster: /base_to_head  
Average rate: 10.190 Hz  
Most recent transform: -0.027 sec old  
Buffer length: 4.907 sec" --> head_frame["/head_frame"]
    head_frame -- "Broadcaster: /HeadMotorHandler  
Average rate: 50.201 Hz  
Most recent transform: 0.015 sec old  
Buffer length: 4.960 sec" --> kinect_normal_axis_frame["/kinect_normal_axis_frame"]
    head_frame -- "Broadcaster: /head_to_arm  
Average rate: 10.196 Hz  
Most recent transform: -0.014 sec old  
Buffer length: 4.806 sec" --> arm_frame["/arm_frame"]
    kinect_normal_axis_frame -- "Broadcaster: /kinect_normal_axis_to_kinect  
Average rate: 10.190 Hz  
Most recent transform: -0.072 sec old  
Buffer length: 4.907 sec" --> kinect_frame["/kinect_frame"]
    
```

例如，当下游的Planning节点想使用Obstacle信息时，需要将Obstacle信息转化到同一个世界坐标系，这时候它会发起一个TF去查询Obstacle处于哪一个世界坐标系里面的哪一个位置，从而感知整个车身周围的情况，基于此再做一个合理的规划和决策行为。

TF Transformation System Tools

Command line

Print information about the current transform tree

```
> rosrn tf tf_monitor
```

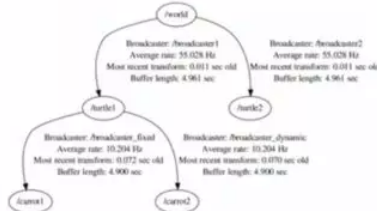
Print information about the transform between two frames

```
> rosrn tf tf_echo
source_frame target_frame
```

View Frames

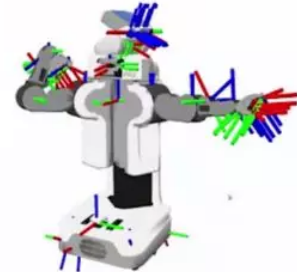
Creates a visual graph (PDF) of the transform tree

```
> rosrn tf view_frames
```



RViz

3D visualization of the transforms



ROS也提供了一些基本功能查看TF的运行机制。例如Rosrun提供了TF监控节点，通过这个节点我们能看整个复杂网络拓扑结构里面节点之间的关系；每个节点之间进行TF转换时所用到的TF树的结构。此外，还提供了tf_echo命令，可以打印从A节点到B节点，例如Perception节点到Planning节点中间所使用了TF转变树的结构。

TF也提供了一个插件供开发者使用，可以用一些主流开发工具进行ROS工程开发。

下图是结合之前将Publiser使用TF的一个例子，想使用TF，只需要改动两部分：第一是定义TF的对象；第二是直接进行TF数据查询，然后就可以得到一个世界坐标系。

TF Transformation System Transform Listener C++ API

- Create a TF listener to fill up a buffer

```
tf2_ros::Buffer tfBuffer;
tf2_ros::TransformListener tfListener(tfBuffer);
```

- Make sure, that the listener does not run out of scope!
- To lookup transformations, use

```
geometry_msgs::TransformStamped transformStamped =
tfBuffer.lookupTransform(target_frame_id,
source_frame_id, time);
```

- For time, use ros::Time(0) to get the latest available transform

```
#include <ros/ros.h>
#include <tf2_ros/transform_listener.h>
#include <geometry_msgs/TransformStamped.h>

int main(int argc, char** argv) {
    ros::init(argc, argv, "tf2_listener");
    ros::NodeHandle nodeHandle;
    tf2_ros::Buffer tfBuffer;
    tf2_ros::TransformListener tfListener(tfBuffer);

    ros::Rate rate(10.0);
    while (nodeHandle.ok()) {
        geometry_msgs::TransformStamped transformStamped;
        try {
            transformStamped = tfBuffer.lookupTransform("base",
                "odom", ros::Time(0));
        } catch (tf2::TransformException &exception) {
            ROS_WARN("%s", exception.what());
            ros::Duration(1.0).sleep();
            continue;
        }
        rate.sleep();
    }
    return 0;
};
```

More info

<http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20listener%20%28C%2B%2B%29>



RQT用户接口

ETH zürich

rqt User Interface

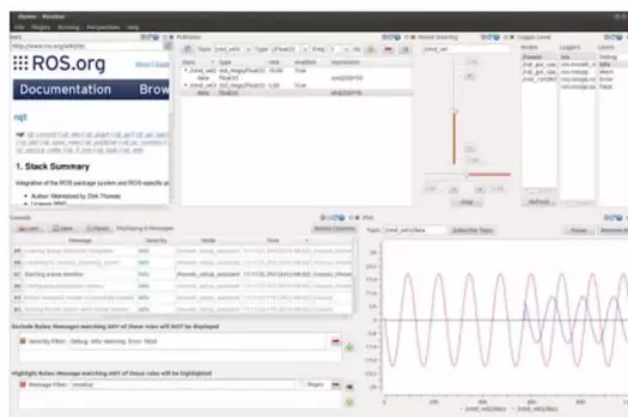
- User interface base on Qt
- Custom interfaces can be setup
- Lots of existing plugins exist
- Simple to write own plugins

Run RQT with

```
> rosrunc rqt_gui rqt_gui
```

or

```
> rqt
```



More info

<http://wiki.ros.org/rqt/Plugins>



RQT顾名思义，R实质是ROS的缩写，QT是可视化的图形工具，RQT是ROS给开发者提供的一套比较方便的图形化相关展示的一套工具。下面介绍几个比较常用的RQT功能：

1. 第一个是RQT imageview，这个主要是为自动驾驶顶层的一些传感器设计的，例如Camera图像，如果你想实时查看Driver接收图像是否正确，颜色、方位是否有问题，可以通过此工具简单选择对应Camera的channel，用 Camera的topic信息实时查看图像状态。

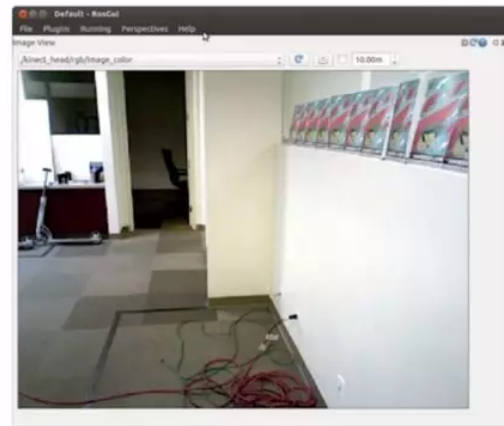
rqt User Interface

rqt_image_view

- Visualizing images

Run *rqt_graph* with

```
> rosrunc rqt_image_view rqt_image_view
```



More info
http://wiki.ros.org/rqt_image_view

2. Multipot可以将二维的数据在一个二维坐标系里面进行实时展示，这样可以更直观地看到我们需要的数据是否符合我们的预期。

3. RQT的graph工具，在开发的实际过程当中使用得比较广泛，这个工具把整个网络拓扑用图形化的方式展现出来。例如启动Perception、Planning和Roscore这三个节点，它都会在RQT graph工具里面进行实时展示，同时两个节点之间所用的topic信息也会在里面实时展示。

rqt User Interface

rqt_graph

- Visualizing the ROS computation graph

Run *rqt_graph* with

```
> rosrunc rqt_graph rqt_graph
```



More info
http://wiki.ros.org/rqt_graph

4. RQT console是对应ROS日志系统所提供的一套可视化工具。ROS提供了五种级别的LOG展示：DEBUG、INFO、WARN、ERROR、FATAL。每个模块在某一时刻都会产生大量的日志信息，RQT

console把这些信息统一规整到一个可视化工具里面，用户可以通过配置的方式快速定位和找到自己所需要的一些相关信息。

ETH zürich

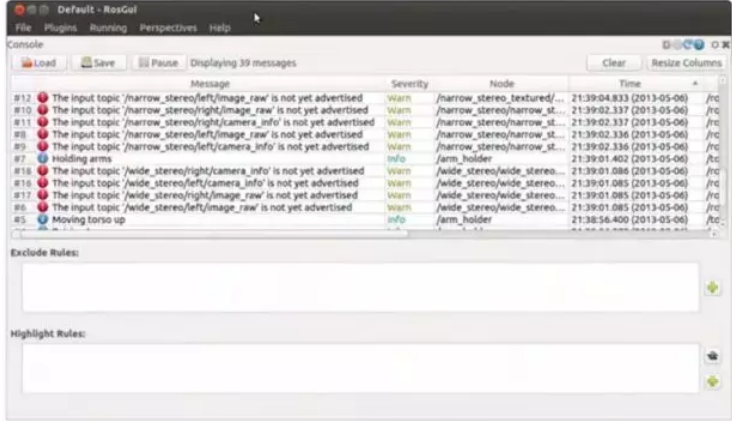
rqt User Interface

rqt_console

- Displaying and filtering ROS messages

Run *rqt_console* with

```
> rosrn rqt_console rqt_console
```



More info
http://wiki.ros.org/rqt_console

5. RQT logleve是为ROS日志系统所提供的另外一个可视化工具。在写代码的时候，可能5种类型的日志都会使用，但是在实际调试过程中可能只想看到某几种类别的实时信息，通过这个工具可以实时调整，让节点输出我们想要的级别的一些信息。例如我们只想看到ERROR或者FATAL信息，就可以把某一个节点的信息级别设置为ERROR，这样这个节点所打印的ERROR和FATAL的信息可以通过命令行或者LOG文件里面去看到，其它级别的信息不会干扰实时调试。

ETH zürich

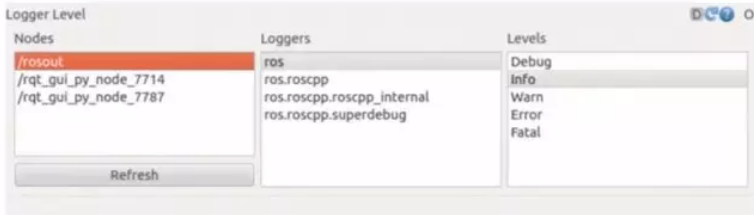
rqt User Interface

rqt_logger_level

- Configuring the logger level of ROS nodes

Run *rqt_logger_level* with

```
> rosrn rqt_logger_level rqt_logger_level
```



More info
http://wiki.ros.org/rqt_logger_level



Robot Models URDF

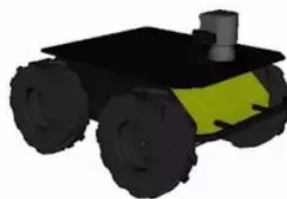
在进行实际模拟的时候，可以用一套语言来定义机器人模型，这套语言被定义为统一机器人描述格式语言URDF。它也是一套xml的语言描述，这个描述格式里面包含两个核心的概念：一个是节点Link，一个是节点之间的连接关系Joint。Joint会指定Parent节点和Child节点，这样就可以描述一个完整的拓扑结构，也就是对整个网络拓扑结构的xml语言化描述。在进行仿真的时候，通过加载对应的URDF文件，在仿真环境里面实时地展示所需要调试的信息。

ETH zürich

Robot Models

Unified Robot Description Format (URDF)

- Defines an XML format for representing a robot model
 - Kinematic and dynamic description
 - Visual representation
 - Collision model
- URDF generation can be scripted with XACRO



Mesh for visuals



Primitives for collision

More info

<http://wiki.ros.org/urdf>

<http://wiki.ros.org/xacro>



SDF Simulation Description Format

Simulation Description Format (SDF) 是另外一个调试工具。之前介绍的Rviz调试工具，更多的是看到消息收发之间的实体化展示，例如展示点云、图像和其它一些信息。如果进行仿真模拟，如机器人模拟的时候，就用另一套工具Gazebo。Gazebo是ROS的一个开发包，它里面所使用的描述语言就是Simulation Description Format。用Gazebo加载URDF时，Gazebo首先把URDF描述语言转换成SDF语言，然后再进行加载和展示。

ETH zürich

Simulation Descriptions

Simulation Description Format (SDF)

- Defines an XML format to describe
 - Environments (lighting, gravity etc.)
 - Objects (static and dynamic)
 - Sensors
 - Robots
- SDF is the standard format for Gazebo
- Gazebo converts a URDF to SDF automatically



More info

<http://sdformat.org>