第五章、ROS常用组件

- TF坐标变换，实现不同类型的坐标系之间的转换；
- rosbag 用于录制ROS节点的执行过程并可以重放该过程；
- rqt 工具箱，集成了多款图形化的调试工具。

可实现的案例：

```
roslaunch turtle_tf2 turtle_tf2_demo_cpp.launch
roslaunch turtle_tf2 turtle_tf2_demo.launch
```

一、TF坐标变换

- 概念：一般是右手坐标系
- tf2常用功能包

```
tf2_geometry_msgs:可以将ROS消息转换成tf2消息
tf2:封装了坐标变换的常量消息
tf2_ros:为tf2提供了roscpp和rospy绑定，封装坐标变换常用api
```

http://wiki.ros.org/tf2

1.坐标msg消息

订阅发布模型中数据载体 msg 是一个重要实现，首先需要了解一下，在坐标转换实现中常用的 msg: `geometry_msgs/TransformStamped` （坐标系位置关系）和 `geometry_msgs/PointStamped` （坐标点消息）

前者用于传输坐标系相关位置信息，后者用于传输某个坐标系内坐标点的信息。在坐标变换中，频繁的需要使用到坐标系的相对关系以及坐标点信息。

1. *geometry_msgs/TransformStamped* ：坐标系位置关系

```
std_msgs/Header header                      #头信息
  uint32 seq                                #|-- 序列号
  time stamp                                #|-- 时间戳
  string frame_id                           #|-- 基坐标名称
string child_frame_id                       #子坐标的名称
geometry_msgs/Transform transform           #坐标信息
  geometry_msgs/Vector3 translation         #偏移量
    float64 x                               #|-- X 方向的偏移量
    float64 y                               #|-- Y 方向的偏移量
    float64 z                               #|-- Z 方向上的偏移量
  geometry_msgs/Quaternion rotation         #四元数
    float64 x
    float64 y
    float64 z
```

```
    float64 w
```

2. *geometry_msgs/PointStamped*：坐标点的位置信息

```
std_msgs/Header header
  uint32 seq    #|-- 序列号
  time stamp    #|-- 时间戳
  string frame_id   #以那个坐标系为参考物的
geometry_msgs/Point point   #具体的坐标位置
  float64 x
  float64 y
  float64 z
```

2.静态坐标变换

所谓静态坐标变换，是指两个坐标系之间的相对位置是固定的。

需求描述:

现有一机器人模型，核心构成包含主体与雷达，各对应一坐标系，坐标系的原点分别位于主体与雷达的物理中心，已知雷达原点相对于主体原点位移关系如下: x 0.2 y0.0 z0.5。当前雷达检测到一障碍物，在雷达坐标系中障碍物的坐标为 (2.0 3.0 5.0),请问，该障碍物相对于主体的坐标是多少?

- 创建功能包

```
catkin_creat_pkg tf01_static roscpp rospy std_msgs tf2 tf2_ros
tf2_geometry_msgs geometry_msgs
```

- 编译一下，看是否抛异常

```
catkin_make
```

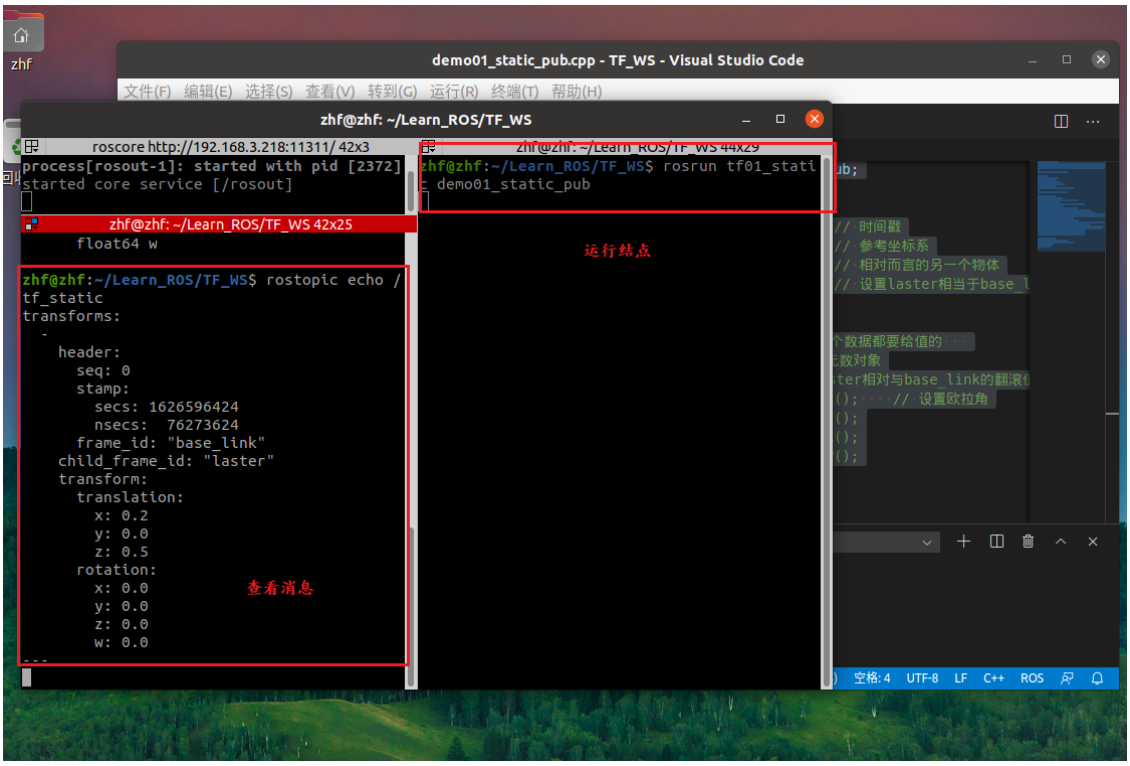
C++实现

- *发布位置信息*

```
#include "ros/ros.h"
#include "tf2/LinearMath/Quaternion.h"                  // 欧拉角转化头文件
#include "tf2_ros/static_transform_broadcaster.h"    // 包含静态坐标对象
#include "geometry_msgs/TransformStamped.h"      // 创建静态坐标消息
/*
    需要：发布两个坐标系的关系
    流程：
        1. 包含头文件
        2. 设置编码
        3. 创建发布对象
        4. 组织被发布的消息
        5. 发布消息
        6. spin();// 发布一次就可以了
*/
int main(int argc, char** argv)
{
    // 2. 设置编码
```

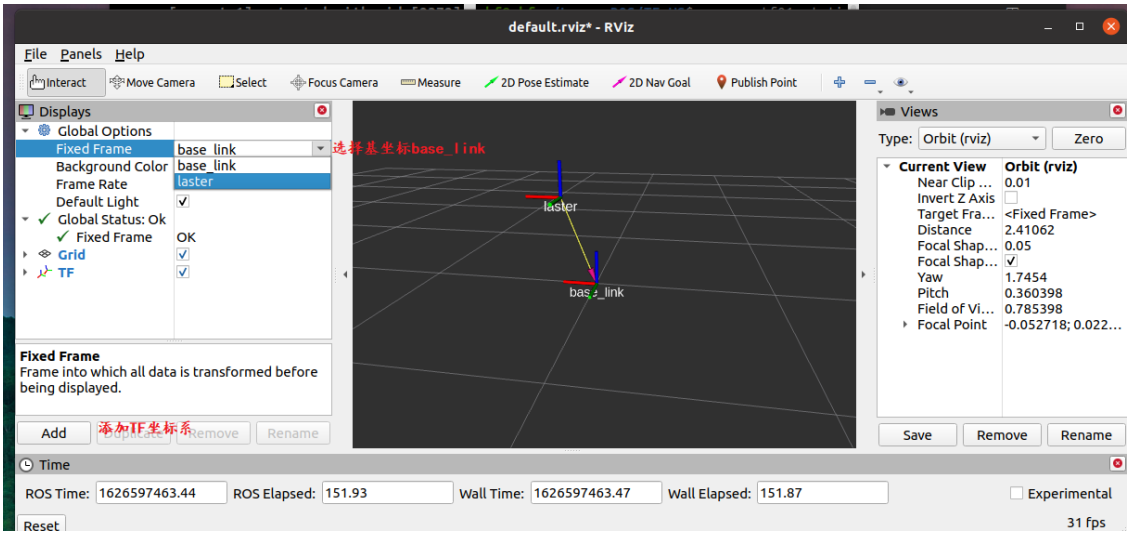
```
18      setlocale(LC_ALL,"");
19      ros::init(argc,argv,"statictf_pub");
20      // 3. 创建静态坐标发布对象
21      tf2_ros::StaticTransformBroadcaster tf_pub;
22      // 4. 组织被发布的消息
23      geometry_msgs::TransformStamped tfs;
24      tfs.header.stamp = ros::Time::now();     // 时间戳
25      tfs.header.frame_id = "base_link";       // 参考坐标系
26      tfs.child_frame_id = "laster";           // 相对而言的另一个物体
27      tfs.transform.translation.x = 0.2;       // 设置laster相当于base_link的
    偏移量
28      tfs.transform.translation.y = 0;
29      tfs.transform.translation.z = 0.5;
30      // 四元组数据有偏移再加也行，并不是消息中的每一个数据都要给值的,但是在rviz中一定
    要加四元数
31      tf2::Quaternion tfqtn;       // 创建四元数对象
32      tfqtn.setRPY(0,0,0);         // 设置laster相对与base_link的翻滚值，俯仰
    值，偏航值。欧拉角的单位是弧度
33      tfs.transform.rotation.x = tfqtn.getX();     // 设置欧拉角
34      tfs.transform.rotation.y = tfqtn.getY();
35      tfs.transform.rotation.z = tfqtn.getZ();
36      tfs.transform.rotation.w = tfqtn.getW();
37
38      // 5. 发布消息
39      tf_pub.sendTransform(tfs);
40      // 6. spin();// 发布一次就可以了
41      ros::spin();
42      return 0;
43  }
44
```

消息的方式查看：



图形化的方式查看：

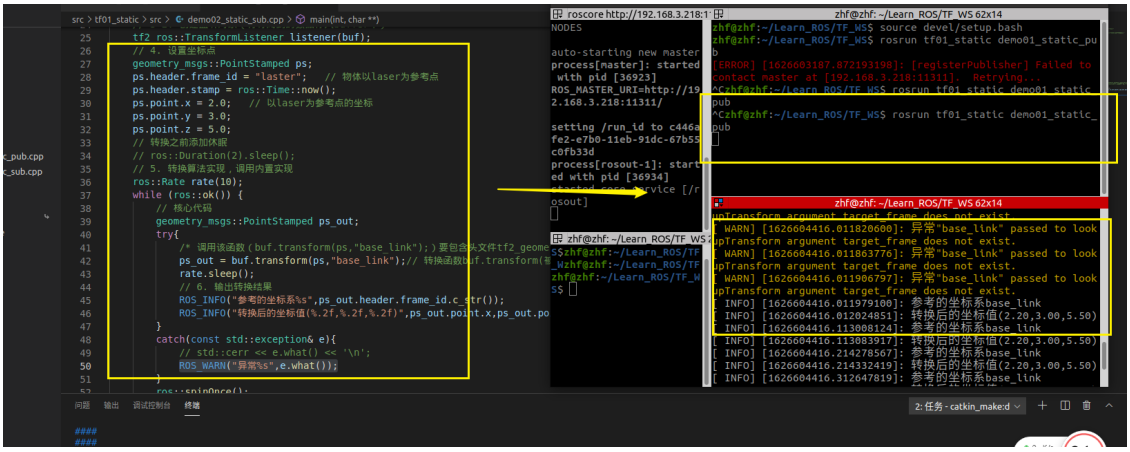
```
# 进入rviz ，直接敲命令rviz
```



- *订阅*

```
#include "ros/ros.h"
#include "tf2_ros/transform_listener.h" // 监听消息头文件，用于创建订阅对象
#include "tf2_ros/buffer.h" // 用于创建数据，存数据
#include "geometry_msgs/PointStamped.h" // 创建坐标点
#include "tf2_geometry_msgs/tf2_geometry_msgs.h"    // 必须加上的一个头文件，否则编译不通过
/*
    订阅方：订阅发布发布消息，传入一个座标点，调用tf进行转换
    流程：
        1. 包含头文件
        2. 编码、初始化、NodeHandle(必须要)
        3. 创建订阅对象
        4. 设置坐标点
        5. 转换算法实现，调用内置实现
        6. 输出转换结果
*/
int main(int argc, char** argv) {
    setlocale(LC_ALL,"");
    // 2. 编码、初始化、NodeHandle(必须要)
    ros::init(argc,argv,"statictf_sub");
    // ros::NodeHandle nh;
    // 3. 创建订阅对象
    // 3-1. 创建buffer缓存
    tf2_ros::Buffer buf;
    // 3-2. 创建监听对象（将订阅到的数据存入buffer中）
    tf2_ros::TransformListener listener(buf);
    // 4. 设置坐标点
    geometry_msgs::PointStamped ps;
    ps.header.frame_id = "laster";   // 物体以laser为参考点
    ps.header.stamp = ros::Time::now();
    ps.point.x = 2.0;    // 以laser为参考点的坐标
    ps.point.y = 3.0;
    ps.point.z = 5.0;
```

```
33      // 转换之前添加休眠
34      // ros::Duration(2).sleep();
35      // 5. 转换算法实现，调用内置实现
36      ros::Rate rate(10);
37      while (ros::ok()) {
38          // 核心代码
39          geometry_msgs::PointStamped ps_out;
40          try{
41              /* 调用该函数（buf.transform(ps,"base_link");）要包含头文件
    tf2_geometry_msgs/tf2_geometry_msgs.h */
42              ps_out = buf.transform(ps,"base_link");// 转换函数
    buf.transform(被转换的坐标点,"基坐标系名称")会返回一个转换完的坐标值。
43              rate.sleep();
44              // 6. 输出转换结果
45              ROS_INFO("参考的坐标系%s",ps_out.header.frame_id.c_str());
46              ROS_INFO("转换后的坐标值
    (%.2f,%.2f,%.2f)",ps_out.point.x,ps_out.point.y,ps_out.point.z);
47          }
48          catch(const std::exception& e){
49              // std::cerr << e.what() << '\n';
50              ROS_WARN("异常%s",e.what());
51          }
52          ros::spinOnce();
53      }
54      return 0;
55  }
```



补、常用组件

1. 用rviz查看坐标关系

```
1  新建窗口输入命令:rviz;
2  在启动的 rviz 中设置Fixed Frame 为 参考坐标系;
3  点击左下的 add 按钮，在弹出的窗口中选择 TF 组件，即可显示坐标关系。
```

2. 直接上命令行

当坐标系之间的相对位置固定时，那么所需参数也是固定的: 父系坐标名称、子级坐标系名称、x偏移量、y偏移量、z偏移量、x 翻滚角度、y俯仰角度、z偏航角度，实现逻辑相同，参数不同，那么ROS 系统就已经封装好了专门的节点，使用方式如下:

```
rosrun tf2_ros static_transform_publisher x偏移量 y偏移量 z偏移量 z偏航角度 y俯仰角度 x翻滚角度 父级坐标系 子级坐标系
```

示例: `rosrun tf2_ros static_transform_publisher 0.2 0 0.5 0 0 0 /baselink /laser`

也建议使用该种方式直接实现静态坐标系相对信息发布。

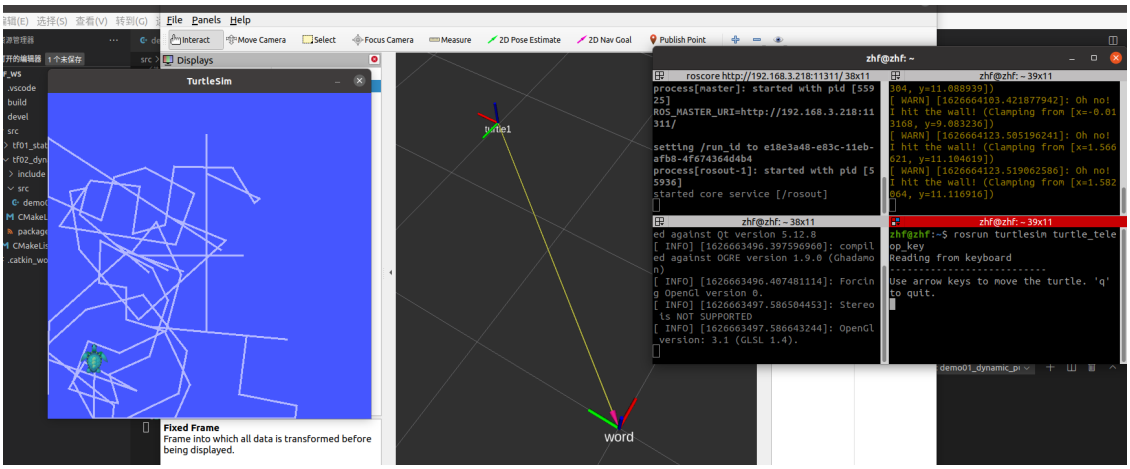
3.动态坐标变换

发布方实现:

```
#include "ros/ros.h"
#include "turtlesim/Pose.h" // 位置消息头文件
#include "tf2_ros/transform_broadcaster.h"  // 包含发布动态坐标关系
#include "geometry_msgs/TransformStamped.h" // tf坐标消息
#include "tf2/LinearMath/Quaternion.h"      // 欧拉角转化头文件
/*
    发布方：需要订阅乌龟的位置信息，转换成相对于窗体的坐标关系并发布
    准 备：
        1. 话题：/turtle1/pose
        2. 消息：turtlesim/Pose
    流程：
        1. 包含头文件；
        2. 设置编码、初始化、NodeHandle；
        3. 创建订阅对象，订阅/turtle1/pose
        4. 回调函数处理订阅的消息：将位置形象转换成相对坐标关系并发布
        5. spin();
*/
void My_do_thing(const turtlesim::Pose::ConstPtr& msg) {
    // 4. 回调函数处理订阅的消息：将位置形象转换成相对坐标关系并发布
    // a. 创建发布对象、创建静态的对象
    static tf2_ros::TransformBroadcaster tf_tur;
    // b. 组织发布数据
    geometry_msgs::TransformStamped ts;
    ts.header.stamp = ros::Time::now();
    ts.header.frame_id = "word";    // 基坐标系
    ts.child_frame_id = "turtle1";  // 乌龟坐标系名称
    ts.transform.translation.x = msg->x;
    ts.transform.translation.y = msg->y;
    ts.transform.translation.z = 0;
    // 坐标系四元数，
    /* 位置信息中没有四元数，但是有偏航角度 */
    tf2::Quaternion qtn;
    qtn.setRPY(0,0,msg->theta);
    ts.transform.rotation.x = qtn.getX();
    ts.transform.rotation.y = qtn.getY();
    ts.transform.rotation.z = qtn.getZ();
    ts.transform.rotation.w = qtn.getW();
    // c. 发布数据
```

```
    tf_tur.sendTransform(ts);
}
int main(int argc, char** argv) {
    // 2. 设置编码、初始化、NodeHandle：
    ros::init(argc,argv,"dynamic_pub");
    ros::NodeHandle nh;
    ros::Subscriber sub = nh.subscribe<turtlesim::Pose>
("/turtle1/pose",100,My_do_thing);
    // 5. spin();
    ros::spin();
    return 0;
}
```

效果：



订阅方实现： 和静态实现差不多

```
#include "ros/ros.h"
#include "tf2_ros/transform_listener.h" // 监听消息头文件，用于创建订阅对象
#include "tf2_ros/buffer.h" // 用于创建数据，存数据
#include "geometry_msgs/PointStamped.h" // 创建坐标点
#include "tf2_geometry_msgs/tf2_geometry_msgs.h"    // 必须加上的一个头文件，否则编译不通过
/*
    订阅方：订阅发布发布消息，传入一个座标点，调用tf进行转换
    流程：
        1. 包含头文件
        2. 编码、初始化、NodeHandle(必须要)
        3. 创建订阅对象
        4. 设置坐标点
        5. 转换算法实现，调用内置实现
        6. 输出转换结果
*/
int main(int argc, char** argv) {
    setlocale(LC_ALL,"");
    // 2. 编码、初始化、NodeHandle(必须要)
    ros::init(argc,argv,"dynamic_sub");
    // ros::NodeHandle nh;
    // 3. 创建订阅对象
    // 3-1. 创建buffer缓存
    tf2_ros::Buffer buf;
```

```
    // 3-2. 创建监听对象（将订阅到的数据存入buffer中）
    tf2_ros::TransformListener listener(buf);
    // 4. 设置坐标点
    geometry_msgs::PointStamped ps;
    ps.header.frame_id = "turtle1";   // 物体以laser为参考点
    ps.header.stamp = ros::Time(0.0);   // 时间需要修改
    ps.point.x = 2.0;   // 以laser为参考点的坐标
    ps.point.y = 3.0;
    ps.point.z = 0.0;
    // 转换之前添加休眠
    ros::Duration(2).sleep();
    // 5. 转换算法实现，调用内置实现
    ros::Rate rate(10);
    while (ros::ok()) {
        // 核心代码
        geometry_msgs::PointStamped ps_out;
        try{
            /* 调用该函数（buf.transform(ps,"base_link");）要包含头文件 tf2_geometry_msgs/tf2_geometry_msgs.h */
            ps_out = buf.transform(ps,"world");// 转换函数buf.transform(被转换的坐标点,"基坐标系名称")会返回一个转换完的坐标值。
            rate.sleep();
            // 6. 输出转换结果
            ROS_INFO("参考的坐标系%s",ps_out.header.frame_id.c_str());
            ROS_INFO("转换后的坐标值(%.2f,%.2f,%.2f)",ps_out.point.x,ps_out.point.y,ps_out.point.z);
        }
        catch(const std::exception& e){
            // std::cerr << e.what() << '\n';
            ROS_WARN("异常%s",e.what());
        }
        ros::spinOnce();
    }
    return 0;
}
```

4.多坐标变换

需求描述：

> 现有坐标系统，父级坐标系统 world,下有两子级系统 son1，son2，son1 相对于 world，以及 son2 相对于 world 的关系是已知的，求 son1原点在 son2中的坐标，又已知在 son1 中一点的坐标，要求求出该点在 son2 中的坐标

1. 创建功能包并导入依赖
2. 创建发布方，发布两个静态坐标

先创建launch文件夹，在文件夹中创建.launch文件。

```xml
<launch>
    <!-- pkg:包名 type:节点类型 name:话题名 args:"偏移量x,y,z,翻转值x,y,z 基坐标 坐标" output="screen" -->
    <node pkg="tf2_ros" type="static_transform_publisher" name="son1" args="5 0 0 0 0 0 /world /son1" output="screen" />
    <node pkg="tf2_ros" type="static_transform_publisher" name="son2" args="3 0 0 0 0 0 /world /son2" output="screen" />
</launch>
```

实现:

```cpp
#include "ros/ros.h"
#include "tf2_ros/transform_listener.h" // 创建接收
#include "tf2_ros/buffer.h"     // 导入buf
#include "geometry_msgs/PointStamped.h"
#include "tf2_geometry_msgs/tf2_geometry_msgs.h"    // 创建点所需头文件
#include "geometry_msgs/TransformStamped.h"
/*
    订阅方实现： 1. 计算son1和son2的相对关系，2.计算son1中的某个点在son2中的坐标值
    流程:
        1. 包含头文件
        2. 编码、初始化、NodeHandle创建
        3. 创建订阅对象
        4. 编写解析逻辑
        5. spinOnce();

*/
int main(int argc, char** argv)
{
    // 2. 编码、初始化、NodeHandle创建
    setlocale(LC_ALL,"");
    ros::init(argc,argv,"demo01_tfs_sub");
    ros::NodeHandle nh;
    // 3. 创建订阅对象
    tf2_ros::Buffer buffer;
    tf2_ros::TransformListener sub(buffer);
    // 4. 编写解析逻辑
    // 坐标点
    geometry_msgs::PointStamped ps_son1;
    ps_son1.header.stamp = ros::Time::now();
    ps_son1.header.frame_id = "son1";
    ps_son1.point.x = 1.0;
    ps_son1.point.y = 2.0;
    ps_son1.point.z = 3.0;
    ros::Rate rate(10);
    while (ros::ok()){
        try{
            // 1. 计算son1与son2的相对关系
            /*
                geometry_msgs::TransformStamped lookupTransform(const std::string& target_frame, const std::string& source_frame,const ros::Time& time) const;
```

```
                参数1const std::string& target_frame：目标坐标系     要当参考坐标系
                参数2const std::string& source_frame：源坐标系       另外一个点
                参数3const ros::Time& time：时间（固定写法）ros::Time(0)
                返回值geometry_msgs::TransformStamped：源相对于目标坐标系的位置关系
            */
            geometry_msgs::TransformStamped ps_out1 = buffer.lookupTransform("son2","son1",ros::Time(0));
            ROS_INFO("son1相对与son2的信息：父极：%s,子集：%s",ps_out1.header.frame_id.c_str(),ps_out1.child_frame_id.c_str());
            ROS_INFO("偏移量（%.2f,%.2f,%.2f）",ps_out1.transform.translation.x,ps_out1.transform.translation.y,ps_out1.transform.translation.z);
            // 2. 计算son1中的某个坐标点在son2中的坐标值
            geometry_msgs::PointStamped ps_out2 = buffer.transform(ps_son1,"son2");
            ROS_INFO("坐标点在%s坐标下的值%.2f,%.2f,%.2f",ps_out2.header.frame_id.c_str(),ps_out2.point.x,ps_out2.point.y,ps_out2.point.z);
            rate.sleep();
            // 5. spinOnce();
            ros::spinOnce();
        }
        catch(const std::exception& e){
            ROS_WARN("异常：%s",e.what());
        }
    }
    return 0;
}
```

5.坐标关系查看

1. 准备工作

首先调用`rospack find tf2_tools`查看是否包含该功能包，如果没有，请使用如下命令安装:

```
sudo apt install ros-noetic-tf2-tools
```

2. 使用

启动坐标系广播程序之后，运行如下命令:（在哪个文件下调用就会生成在哪个文件夹下面）

```
rosrun tf2_tools view_frames.py
```

会产生类似于下面的日志信息:

```
[INFO] [1592920556.827549]: Listening to tf data during 5 seconds...
[INFO] [1592920561.841536]: Generating graph in frames.pdf file...
```

查看当前目录会生成一个 frames.pdf 文件

可以直接进入目录打开文件，或者调用命令查看文件:`evince frames.pdf`

6.乌龟跟随案例

1. 创建第二只小乌龟

```
#include "ros/ros.h"
#include "turtlesim/Spawn.h"
int main(int argc, char** argv)
{
    // 设置编码
    setlocale(LC_ALL,"");
    ros::init(argc,argv,"request_spawn");
    ros::NodeHandle nh;
    // 创建客户端对象，发布请求
    ros::ServiceClient client_spawn = nh.serviceClient<turtlesim::Spawn>
("/spawn");
    // 等待请求端
    ros::service::waitForService("/spawn");
    // 创建位置信息
    turtlesim::Spawn sp;
    sp.request.name = "turtle2";
    sp.request.x = 1;
    sp.request.y = 1;
    sp.request.theta = 3.14;
    // 发布请求
    bool flag = client_spawn.call(sp);
    if(flag)
        ROS_INFO("创建成功");
    else
        ROS_INFO("创建失败");
    ros::spin();
    return 0;
}
```

2. 将乌龟位置信息发布到TF坐标系下

```
#include "ros/ros.h"
#include "turtlesim/Pose.h" // 位置消息头文件
#include "tf2_ros/transform_broadcaster.h"  // 包含发布动态坐标关系
#include "geometry_msgs/TransformStamped.h" // tf坐标消息
#include "tf2/LinearMath/Quaternion.h"      // 欧拉角转化头文件
std::string name;
void My_do_thing(const turtlesim::Pose::ConstPtr& msg) {
    // 4. 回调函数处理订阅的消息：将位置形象转换成相对坐标关系并发布
    // a. 创建发布对象、创建静态的对象
    static tf2_ros::TransformBroadcaster tf_tur;
    // b. 组织发布数据
    geometry_msgs::TransformStamped ts;
    ts.header.stamp = ros::Time::now();
    ts.header.frame_id = "world";    // 基坐标系
    ts.child_frame_id = name;  // 乌龟坐标系名称
    ts.transform.translation.x = msg->x;
    ts.transform.translation.y = msg->y;
    ts.transform.translation.z = 0;
```

```
    // 坐标系四元数，
    /* 位置信息中没有四元数，但是有偏航角度 */
    tf2::Quaternion qtn;
    qtn.setRPY(0,0,msg->theta);
    ts.transform.rotation.x = qtn.getX();
    ts.transform.rotation.y = qtn.getY();
    ts.transform.rotation.z = qtn.getZ();
    ts.transform.rotation.w = qtn.getW();
    // c. 发布数据
    tf_tur.sendTransform(ts);
}
int main(int argc, char** argv) {
    // 2. 设置编码、初始化、NodeHandle；
    setlocale(LC_ALL,"");
    ros::init(argc,argv,"send_tf");
    if (argc!=2)
    {
        ROS_ERROR("参数错误！！！");
    }else{
        name = argv[1];
        ROS_INFO("成功。");
    }
    ros::NodeHandle nh;
    ros::Subscriber sub = nh.subscribe<turtlesim::Pose>
(name+"/pose",100,My_do_thing);
    // 5. spin();
    ros::spin();
    return 0;
}
```

3. 设置线速度和角速度

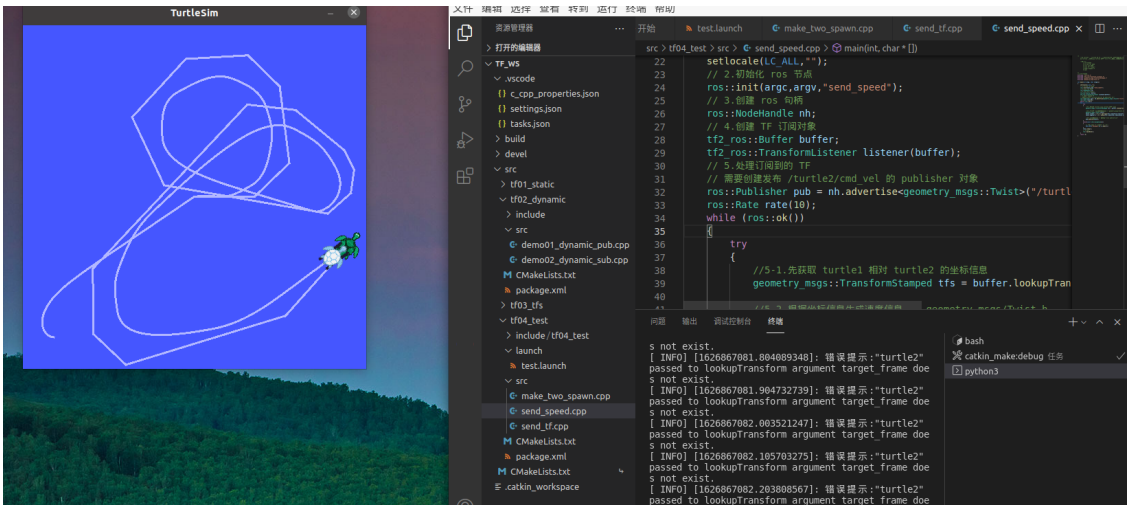
```
/*
    订阅 turtle1 和 turtle2 的 TF 广播信息，查找并转换时间最近的 TF 信息
    将 turtle1 转换成相对 turtle2 的坐标，在计算线速度和角速度并发布
    实现流程:
        1.包含头文件
        2.初始化 ros 节点
        3.创建 ros 句柄
        4.创建 TF 订阅对象
        5.处理订阅到的 TF
        6.spin
*/
//1.包含头文件
#include "ros/ros.h"
#include "tf2_ros/transform_listener.h"
#include "geometry_msgs/TransformStamped.h"
#include "geometry_msgs/Twist.h"

int main(int argc, char *argv[])
{
    setlocale(LC_ALL,"");
```

```cpp
    // 2.初始化 ros 节点
    ros::init(argc,argv,"send_speed");
    // 3.创建 ros 句柄
    ros::NodeHandle nh;
    // 4.创建 TF 订阅对象
    tf2_ros::Buffer buffer;
    tf2_ros::TransformListener listener(buffer);
    // 5.处理订阅到的 TF
    // 需要创建发布 /turtle2/cmd_vel 的 publisher 对象
    ros::Publisher pub = nh.advertise<geometry_msgs::Twist>
("/turtle2/cmd_vel",1000);
    ros::Rate rate(10);
    while (ros::ok())
    {
        try
        {
            //5-1.先获取 turtle1 相对 turtle2 的坐标信息
            geometry_msgs::TransformStamped tfs =
buffer.lookupTransform("turtle2","turtle1",ros::Time(0));

            //5-2.根据坐标信息生成速度信息 -- geometry_msgs/Twist.h
            geometry_msgs::Twist twist;
            twist.linear.x = 0.5 *
sqrt(pow(tfs.transform.translation.x,2) +
pow(tfs.transform.translation.y,2));
            twist.angular.z = 4 *
atan2(tfs.transform.translation.y,tfs.transform.translation.x);

            //5-3.发布速度信息 -- 需要提前创建 publish 对象
            pub.publish(twist);
        }
        catch(const std::exception& e)
        {
            // std::cerr << e.what() << '\n';
            ROS_INFO("错误提示:%s",e.what());
        }
        rate.sleep();
        // 6.spin
        ros::spinOnce();
    }
    return 0;
}
```



二、rosbag

> 机器人导航实现中，可能需要绘制导航所需的全局地图，地图绘制实现，有两种方式，方式1：可以控制机器人运动，将机器人传感器感知到的数据时时处理，生成地图信息。方式2：同样是控制机器人运动，将机器人传感器感知到的数据留存，事后，再重新读取数据，生成地图信息。两种方式比较，显然方式2使用上更为灵活方便。

概念： 用于录制和回放ROS主题的一个工具。

作用： 数据的复用，方便调试、测试。

本质： rosbag本质也是ros的节点，当录制时，rosbag是一个订阅节点，可以订阅话题消息并将订阅到的数据写入磁盘文件；当重放时，rosbag是一个发布节点，可以读取磁盘文件，发布文件中的话题消息。

1.命令行使用

1. 创建文件夹（文件夹位置没有特殊规定，任何位置都可以）保存录制的操作

```
mkdir bags
```

2. 开始录制

```
rosbag record -a -O ~路径/目标文件
```

操作小乌龟一段时间，结束录制使用 ctrl + c，在创建的目录中会生成bag文件。

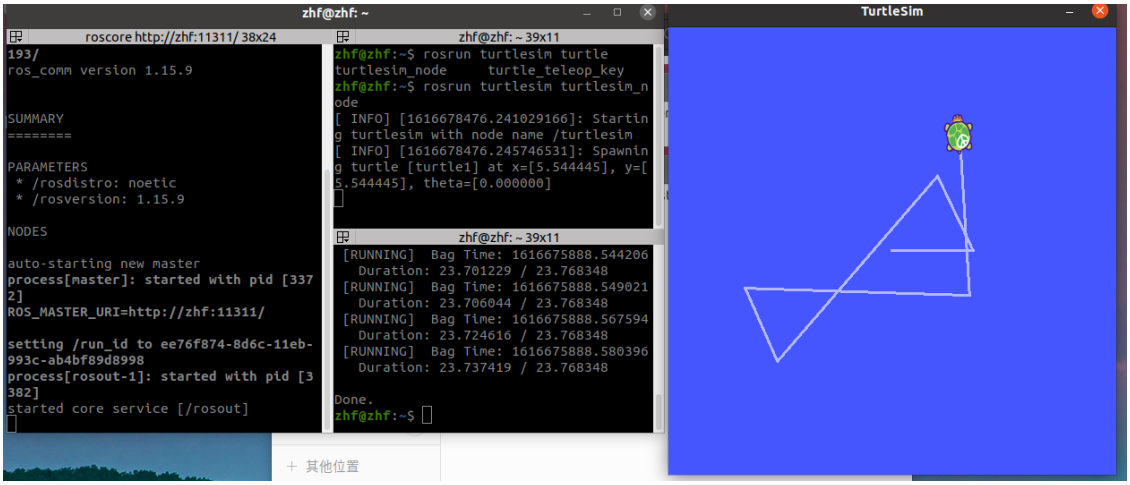
3. 查看文件信息

```
rosbag info 文件名
```

4. 回放文件

```
rosbag play 文件名
```

重启乌龟节点，会发现，乌龟按照录制时的轨迹运动。



2.编码方式

1. 写操作

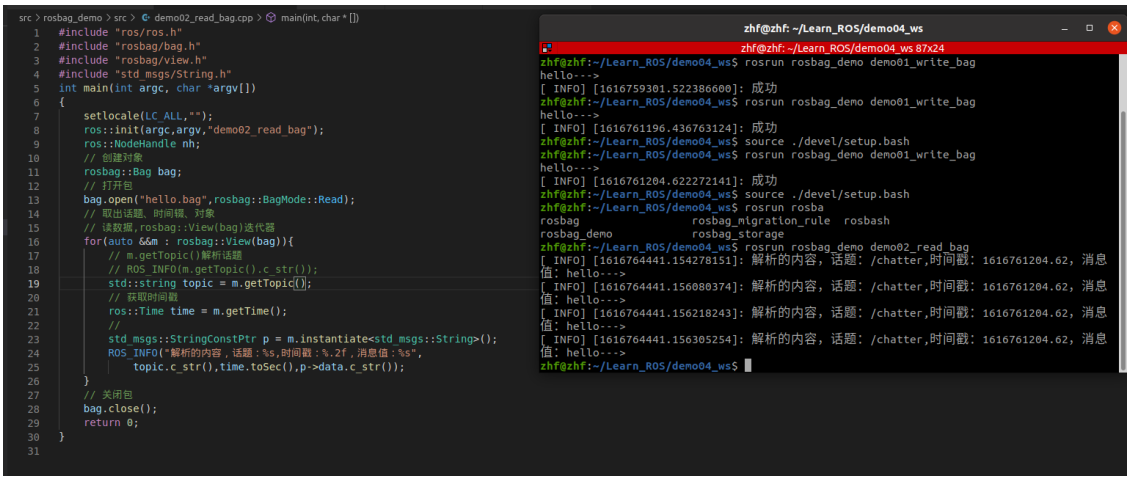
```
#include <ros/ros.h>
// 导入头文件
#include "rosbag/bag.h"
#include "std_msgs/String.h"
int main(int argc, char *argv[]) {
    // 防止中文乱码
    setlocale(LC_ALL,"");
    ros::init(argc,argv,"demo01_write_bag");
    ros::NodeHandle nh;
    // 创建rosbag对象
    rosbag::Bag bag;
    // 打开文件流
    bag.open("hello.bag",rosbag::BagMode::Write);
    // 写数据

    std_msgs::String msg;
    msg.data = "hello--->";
    // 参数一：话题名称，参数二：时间，参数三：消息。
    bag.write("/chatter",ros::Time::now(),msg);
    bag.write("/chatter",ros::Time::now(),msg);
    bag.write("/chatter",ros::Time::now(),msg);
    bag.write("/chatter",ros::Time::now(),msg);
    std::cout << msg.data << std::endl;
    ROS_INFO("成功");
    // 关闭文件流
    bag.close();
    return 0;
}
```

2. 读操作

```
#include "ros/ros.h"
#include "rosbag/bag.h"
#include "rosbag/view.h"
```

```cpp
#include "std_msgs/String.h"
int main(int argc, char *argv[])
{
    setlocale(LC_ALL,"");
    ros::init(argc,argv,"demo02_read_bag");
    ros::NodeHandle nh;
    // 创建对象
    rosbag::Bag bag;
    // 打开包
    bag.open("hello.bag",rosbag::BagMode::Read);
    // 取出话题、时间辍、对象
    // 读数据,rosbag::View(bag)迭代器
    for(auto &&m : rosbag::View(bag)){
        // m.getTopic()解析话题
        // ROS_INFO(m.getTopic().c_str());
        std::string topic = m.getTopic();
        // 获取时间戳
        ros::Time time = m.getTime();
        // 消息值
        std_msgs::StringConstPtr p = m.instantiate<std_msgs::String>();
        ROS_INFO("解析的内容，话题：%s,时间戳：%.2f，消息值：%s",
            topic.c_str(),time.toSec(),p->data.c_str());
    }
    // 关闭包
    bag.close();
    return 0;
}
```



三、rqt工具箱

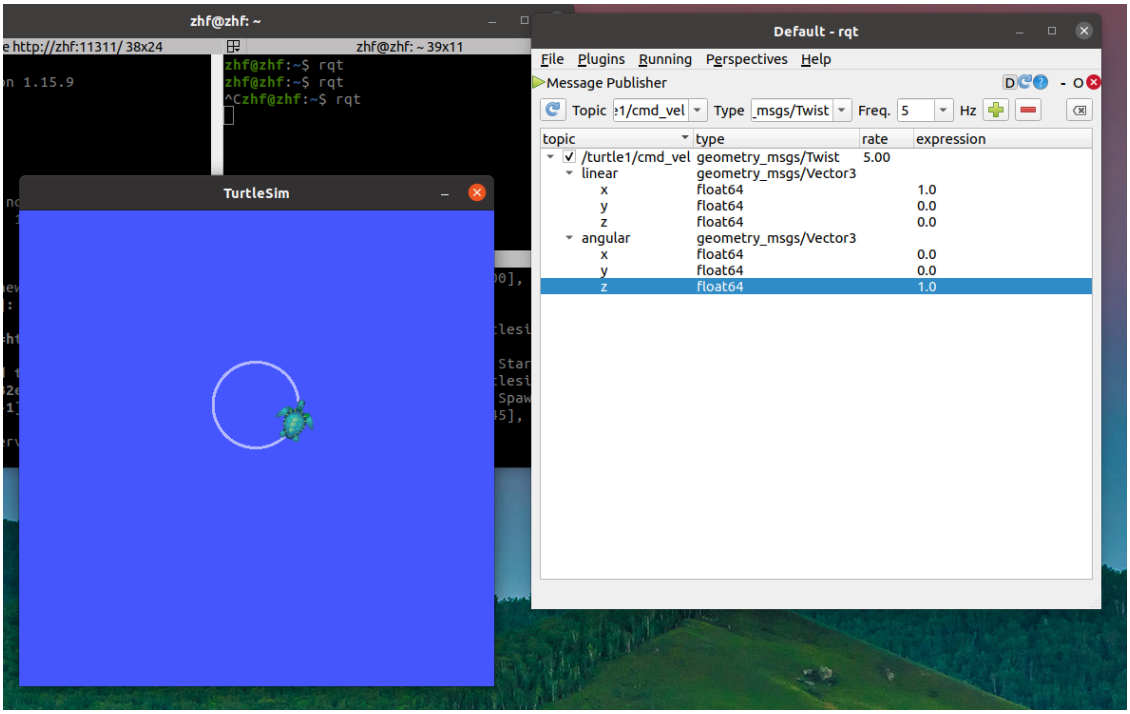
1.安装启动与基本使用

1. 安装(没有安装的话才安装)

```
sudo apt-get install ros-noetic-rqt
sudo apt-get install ros-noetic-rqt-common-plugins
```

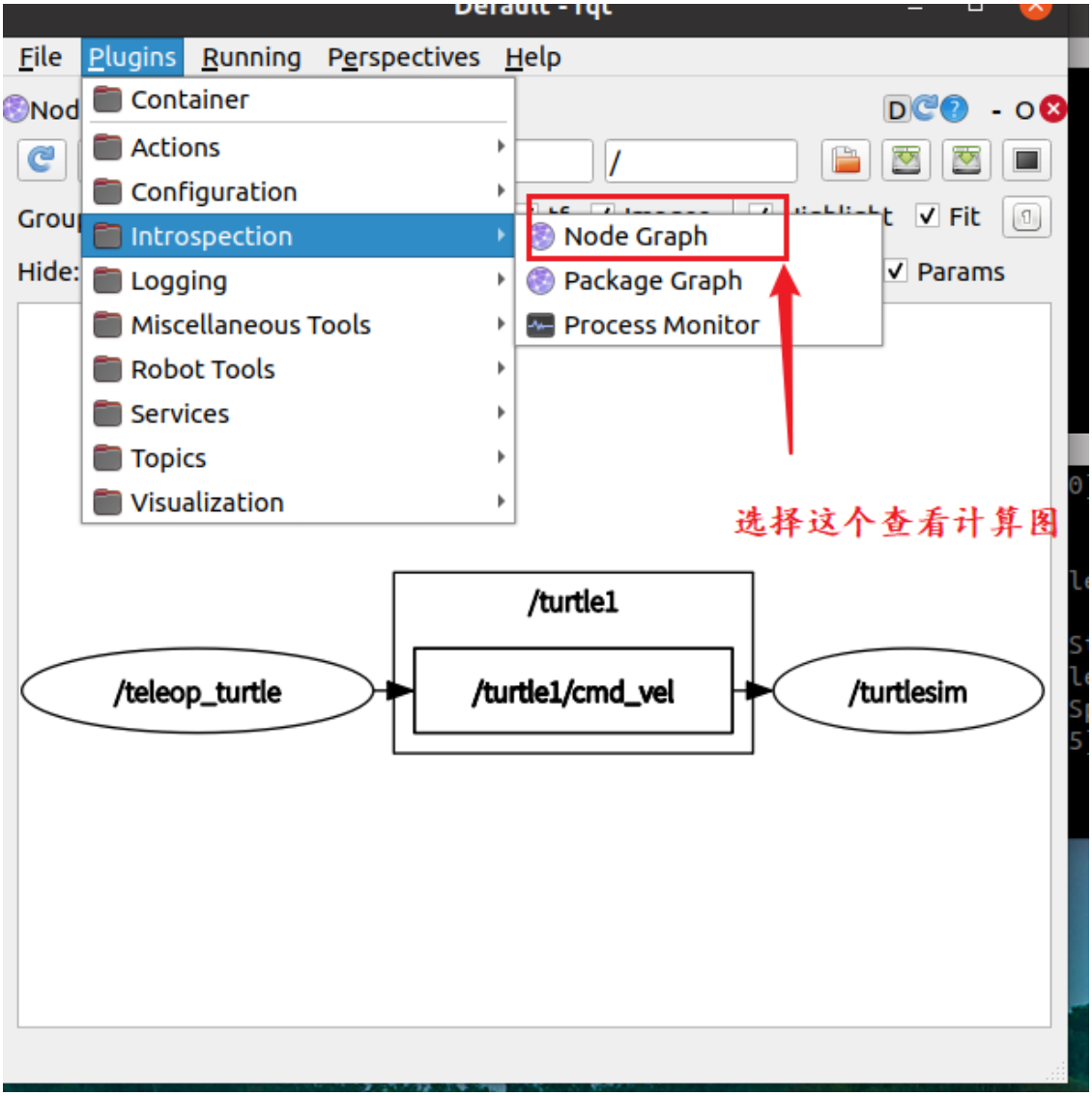
2. 启动

```
1  rqt
```



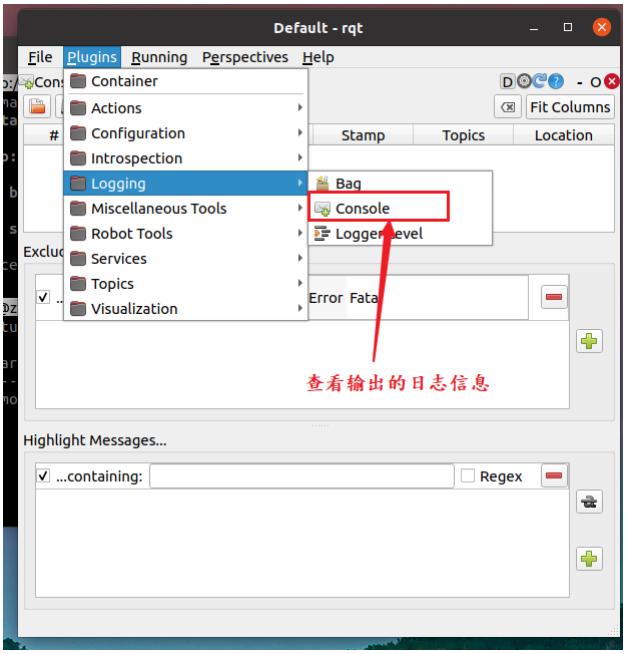
2.rqt插件：rqt_graph(可视化显示计算图)

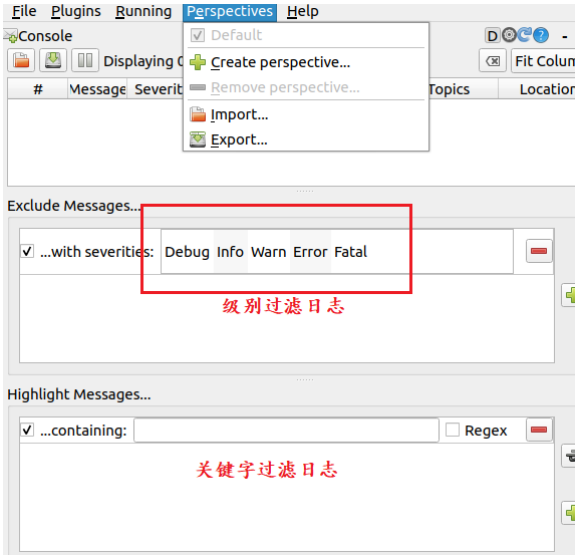
启动: 可以在 rqt 的 plugins 中添加，或者使用 `rqt_graph` 启动



3.rqt插件：rqt_console(显示和过滤日志)

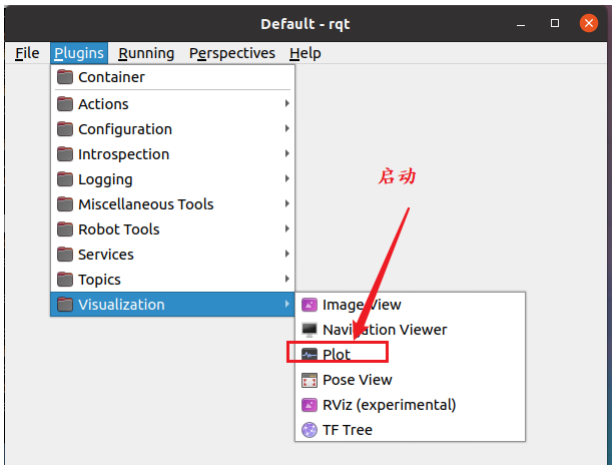
启动：可以在 rqt 的 plugins 中添加，或者使用 `rqt_console` 启动

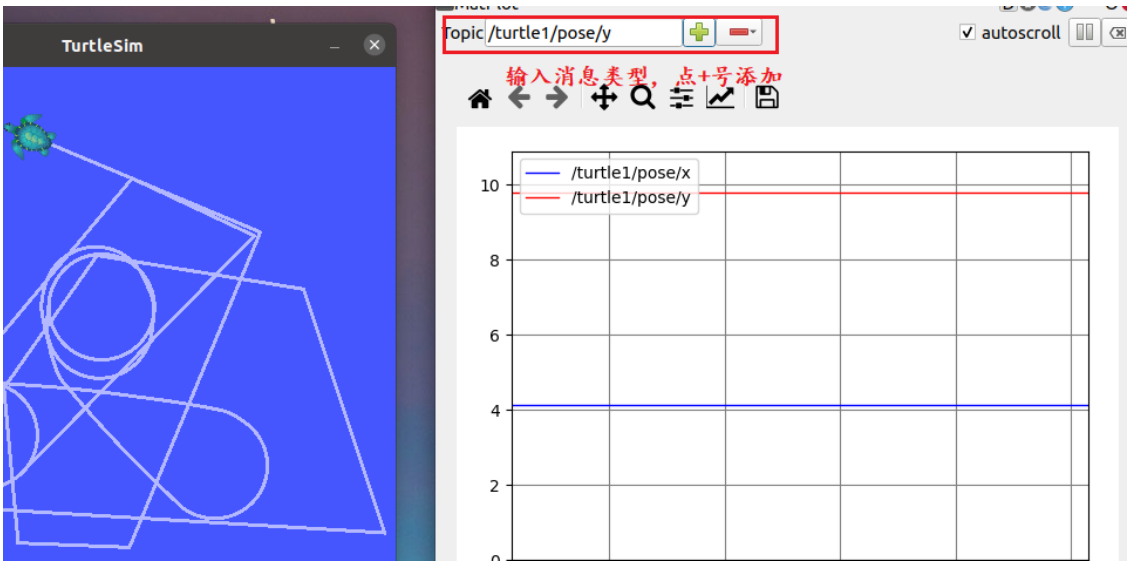




4.rqt插件：rqt_plot(图形绘制)

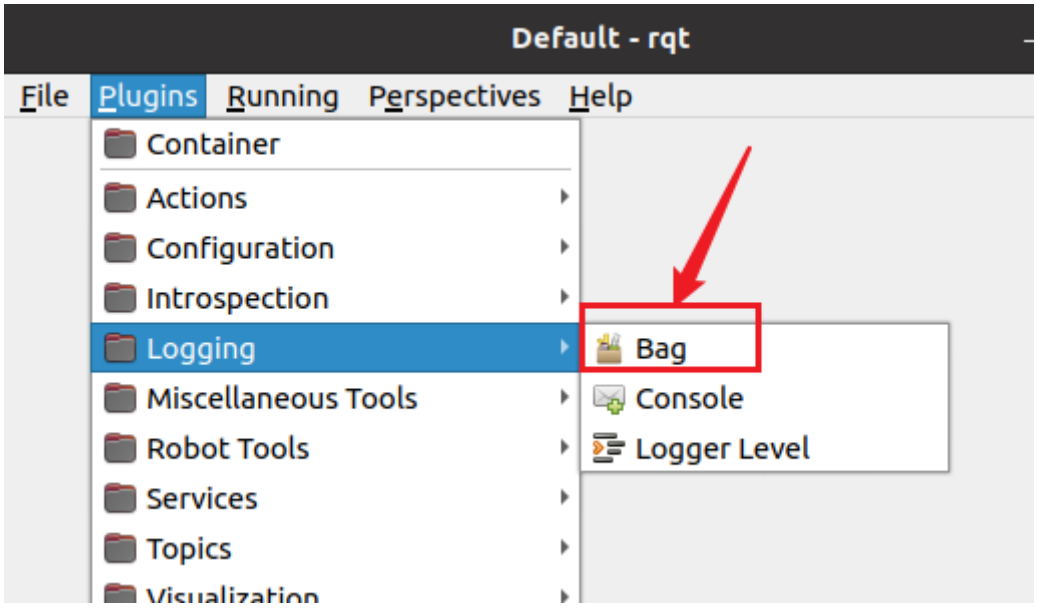
启动: 可以在 rqt 的 plugins 中添加，或者使用 `rqt_plot` 启动



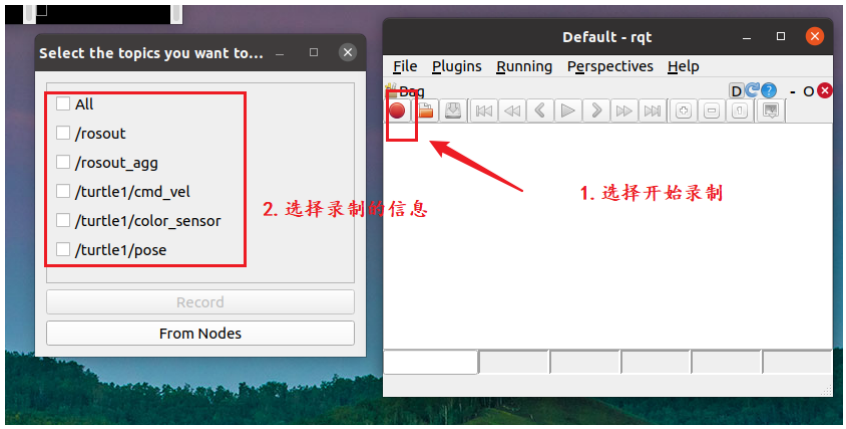


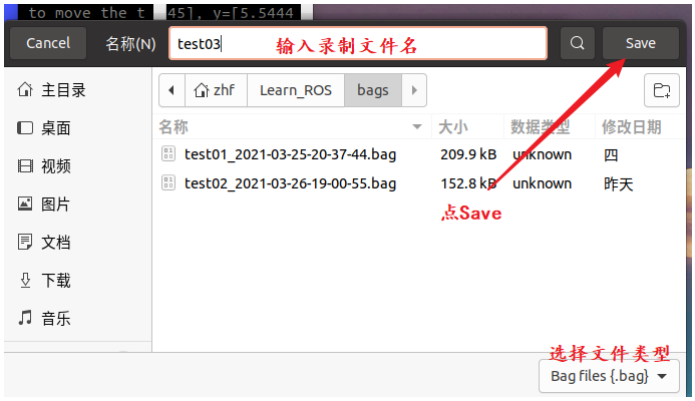
5.rqt插件：rqt_bag(录制和重放 bag 文件)

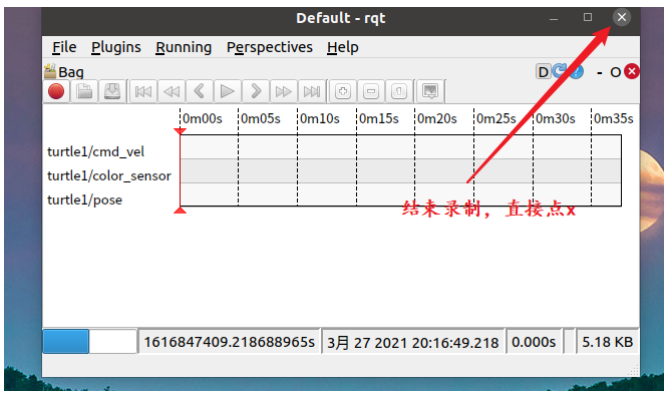
启动: 可以在 rqt 的 plugins 中添加，或者使用 `rqt_bag` 启动

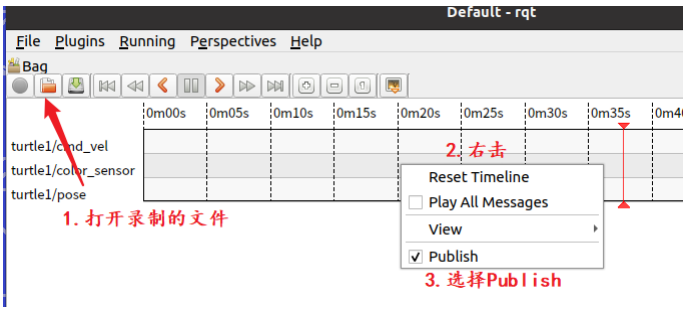

Default - rqt
File
Plugins
Running
Perspectives
Help
Container
Actions
Configuration
Introspection
Logging
Miscellaneous Tools
Robot Tools
Services
Topics
Visualization
Bag
Console
Logger Level

操作:


Select the topics you want to...
All
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
Record
From Nodes
2. 选择录制的信息
Default - rqt
File
Plugins
Running
Perspectives
Help
Bag
1. 选择开始录制


Cancel
名称(N)
test03
输入录制文件名
Save
主目录
桌面
视频
图片
文档
下载
音乐
zhf
Learn_ROS
bags
名称
大小
数据类型
修改日期
test01_2021-03-25-20-37-44.bag
209.9 kB
unknown
四
test02_2021-03-26-19-00-55.bag
152.8 kB
unknown
昨天
点Save
选择文件类型
Bag files (.bag)


Default - rqt
File
Plugins
Running
Perspectives
Help
Bag
0m00s
0m05s
0m10s
0m15s
0m20s
0m25s
0m30s
0m35s
turtle1/cmd_vel
turtle1/color_sensor
turtle1/pose
结束录制，直接点x
1616847409.218688965s
3月 27 2021 20:16:49.218
0.000s
5.18 KB


Default - rqt
File Plugins Running Perspectives Help
Bag
0m00s 0m05s 0m10s 0m15s 0m20s 0m25s 0m30s 0m35s
turtle1/cmd_vel
turtle1/color_sensor
turtle1/pose
1. 打开录制的文件
2. 右击
Reset Timeline
Play All Messages
View
Publish
3. 选择Publish