概率机器人三级项目

概述：

机器人是一种高度复杂的系统性实现，机器人设计包含了机械结构设计、机械加工、硬件设计、嵌入式软件设计、上层软件设计等诸多模块，是各种硬件与软件的有机结合，甚至可以说机器人系统是当今工业体系的集大成者。机器人体系是相当庞大的，其复杂度之高，以至于没有任何个人、组织甚至公司能够独立完成系统性的机器人研发生产任务。

因此，为了能让让机器人研发者专注于自己擅长的领域，其他模块则直接复用相关领域更专业研发团队的实现，当然自身的研究也可以被他人继续复用，ROS应运而生。ROS是一套机器人通用软件框架，可以提升功能模块的复用性，并且随着ROS2的推出，ROS日臻完善，是机器人软件开发的不二之选。

自ROS诞生的十几年来，不管是机器人相关软件、硬件还是ROS社区都发生了天翻地覆的变化，加之ROS1存在一些设计上的先天性缺陷，各种内外因素叠加下，导致ROS1在许多应用场景下都已经显得力不从心了。此背景下，官方于2017正式推出了新一代机器人操作系统——ROS2，ROS2基于全新的设计框架，保留了ROS1的优点并改进其缺陷，ROS2的目标是适应新时代的新需求。

所以，本次三级项目我组采用ROS2来完成topic 通讯程序编写以及TF 广播器和

监听器的编程实现。

正文：

题目一：控制乌龟做圆形运动熟悉 ROS 当中的 topic 通讯，并自主编写控制程序，实现乌龟的矩形运动。

ROS的话题（Topics）通信机制是基于发布/订阅模型实现的异步通信方式，用于在ROS节点之间传输消息。它是ROS中最常用的通信机制之一，非常适合实时数据的传输。

在ROS中，节点可以发布（Publish）消息到一个话题，并且其他节点可以订阅（Subscribe）该话题来接收消息。发布者和订阅者之间是松耦合的，它们不需要知道对方的存在，只需要关注同一个话题名字即可。

对于如何控制乌龟做矩形运动，首先需要创建一个节点作为发布方，向话题’turtle1/cmd\_vel’发布<geometry\_msgs::msg::Twist>类型的消息，其中，’turtle1/cmd\_vel’为与乌龟运动相关的话题，<geometry\_msgs::msg::Twist>为话题中运动消息的类型。

然后ROS2可以用节点创建一个定时器用于循环发布话题消息。定时器周期进行调用回调函数，内部可利用ROS2节点发布方的方法进行用户自定义的消息发布。

具体而言，为了让乌龟进行矩形运动，首先需要先让乌龟直线运动一段时间(编程中设定count=5)，然后给乌龟有一定的加速度进行专项，循环执行以上操作即可完成乌龟矩形运动。

\*\*使用ROS2编写 Rectangular.cpp\*\*

#include <chrono>

#include <memory>

#include "rclcpp/rclcpp.hpp"

#include "std\_msgs/msg/string.hpp"

#include "geometry\_msgs/msg/twist.hpp"

#define PI 3.14159265358979323846

//topic:turtle1/cmd\_vel message\_type:[geometry\_msgs/msg/Twist]

using namespace std::chrono\_literals;

class MinimalPublisher : public rclcpp::Node

{

public:

MinimalPublisher()//node

: Node("minimal\_publisher"), count\_(0)

{

publisher\_ = this->create\_publisher<geometry\_msgs::msg::Twist>("turtle1/cmd\_vel", 10);//init

timer\_ = this->create\_wall\_timer(

500ms, std::bind(&MinimalPublisher::timer\_callback, this));

}

private:

void timer\_callback()

{

static int count = 0;

geometry\_msgs::msg::Twist speed;

speed.linear.x = 1; // 设置线速度为1m/s，正为前进，负为后退

speed.linear.y = 0;

speed.linear.z = 0;

speed.angular.x = 0;

speed.angular.y = 0;

speed.angular.z = 0;

count++;

while(count == 5)

{

count=0;

speed.linear.x = 1;

speed.linear.y = 0;

speed.linear.z = 0;

speed.angular.x = 0;

speed.angular.y = 0;

speed.angular.z = PI; //转90°

}

publisher\_->publish(speed);

}

rclcpp::TimerBase::SharedPtr timer\_;

rclcpp::Publisher<geometry\_msgs::msg::Twist>::SharedPtr publisher\_;

size\_t count\_;

};

int main(int argc, char \* argv[])

{

rclcpp::init(argc, argv);

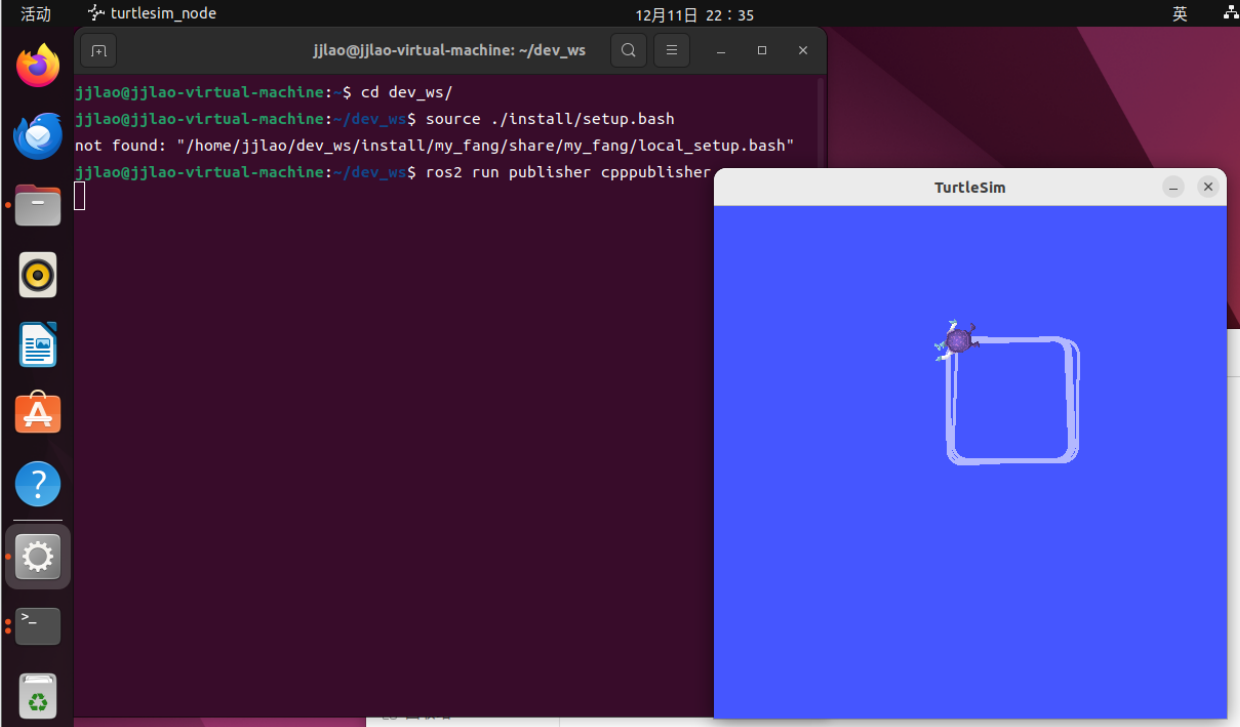
rclcpp::spin(std::make\_shared<MinimalPublisher>());

rclcpp::shutdown();

return 0;

}

代码功能效果如图所示：



图中乌龟成功实现矩形运动。说明代码实现了乌龟运动消息的发布。

题目二：熟悉 ROS 中 TF 坐标变换的实现机制广播与监听的编程实现，并自主编写代码实现 TF 广播器和 监听器，实现乌龟跟随移动。

ROS2内部封装实现了TF 坐标变换的实现机制广播与监听的相关方法函数。为了调用这些函数完成题目二，我们需要理解ROS2的参数服务功能、TF广播器、TF监听器。

首先，实现乌龟跟随移动需要两只乌龟，其中一只乌龟1可以调用ros2 run turtlesim turtle\_teleop\_key来控制乌龟的移动，另一只乌龟2通过接受TF监听器节点作为发布方发布的运动消息，跟随乌龟1的移动。由于ros2 run turtlesim turtlesim\_node调用后只会生成一只乌龟，因此我们需要编写一个程序调用turtlesim相关参数服务，调用spawn生成第二只乌龟，编写内容主要是使用参数服务声明新的乌龟的信息，然后创建服客户端去连接服务端，最后组织发布乌龟信息，即可生成一只新乌龟。

然后我们需要两个节点分别作为TF广播器进行广播两只乌龟相对于word世界坐标系的位姿关系。其中，"tf2\_ros/transform\_broadcaster.h"是创建动态广播器的类型相关的库函数，"geometry\_msgs/msg/transform\_stamped.hpp"是动态广播器能广播出去的话题类型的库函数，"tf2/LinearMath/Quaternion.h"是转换欧拉角--四元数相关库函数。

最后我们需要建立一个节点作为TF监听器，调用监听器方法去监听广播器接受位姿关系，并调用另一个方法自动完成位姿关系计算完成TF坐标变换。最后该节点同样可以创建一个广播器，用于将坐标变换后换算的速度等运动消息，向’turtle2/cmd\_vel’也就是第二只乌龟发布运动消息，即可让乌龟2完成对乌龟1的跟随运动。

\*\*使用ROS2编程 spawn.cpp\*\*

#include <chrono>

#include <memory>

#include "rclcpp/rclcpp.hpp"

#include "turtlesim/srv/spawn.hpp"//

#include "tf2\_ros/transform\_listener.h"//创建监听器的类型

#include "std\_msgs/msg/string.hpp"

//1.使用参数服务声明新的乌龟的信息

//2.创建服务客户端

//3.连接服务端

//4.组织并发送数据

using namespace std::chrono\_literals;

class Spawn : public rclcpp::Node

{

public:

Spawn(): Node("Spawn\_node")

{

//1.使用参数服务声明新的乌龟的信息

this->declare\_parameter("x", 3.0);

this->declare\_parameter("y", 3.0);

this->declare\_parameter("theta", 0.0);

this->declare\_parameter("turtle\_name", "turtle2");

x = this->get\_parameter("x").as\_double();

y = this->get\_parameter("y").as\_double();

theta = this->get\_parameter("theta").as\_double();

turtle\_name = this->get\_parameter("turtle\_name").as\_string();

//2.创建服务客户端

spawn\_client\_ = this->create\_client<turtlesim::srv::Spawn>("/spawn");

}

//3.连接服务端

bool connect\_server()

{

while(!spawn\_client\_->wait\_for\_service(1s))

{

if(!rclcpp::ok())

{

RCLCPP\_INFO(rclcpp::get\_logger("rclcpp"), "强制退出!");

return false;

}

RCLCPP\_INFO(this->get\_logger(), "服务连接中......");

}

return true;

}

//4.组织并发送数据

rclcpp::Client<turtlesim::srv::Spawn>::FutureAndRequestId request()

{

auto req = std::make\_shared<turtlesim::srv::Spawn::Request>();

req->x = x;

req->y = y;

req->theta = theta;

req->name = turtle\_name;

return spawn\_client\_->async\_send\_request(req);

}

private:

double\_t x, y, theta;

std::string turtle\_name;

rclcpp::Client<turtlesim::srv::Spawn>::SharedPtr spawn\_client\_;

};

int main(int argc, char \* argv[])

{

rclcpp::init(argc, argv);

//由于节点任务是发布参数服务，服务完就可以关闭了，因此不需要span函数

//rclcpp::spin(std::make\_shared<TF\_listener>());

//创建自定义节点类对象，组织函数，处理响应结果

auto client\_ = std::make\_shared<Spawn>();

bool flag = client\_->connect\_server();

if(!flag)

{

RCLCPP\_INFO(rclcpp::get\_logger("rclcpp"), "服务连接失败");

return 1;

}

//发送请求

auto response = client\_->request();

//处理响应

if(rclcpp::spin\_until\_future\_complete(client\_, response) == rclcpp::FutureReturnCode::SUCCESS)

{

RCLCPP\_INFO(client\_->get\_logger(), "响应成功！");

std::string name = response.get()->name;

if(name.empty())

{

RCLCPP\_INFO(client\_->get\_logger(), "生成的乌龟因为重名而生成失败！");

}

else

{

RCLCPP\_INFO(client\_->get\_logger(), "生成乌龟成功！");

}

}

else

{

RCLCPP\_INFO(client\_->get\_logger(), "响应失败！");

}

rclcpp::shutdown();

return 0;

}

\*\*使用ROS2编程 tf\_broadcaster.cpp\*\*

#include <chrono>

#include <memory>

#include "rclcpp/rclcpp.hpp"

#include "tf2\_ros/transform\_broadcaster.h"//创建动态广播器的类型

#include "turtlesim/msg/pose.hpp"//订阅乌龟1的pose话题

#include "geometry\_msgs/msg/transform\_stamped.hpp"//动态广播器能广播出去的话题类型

#include "tf2/LinearMath/Quaternion.h"//转换欧拉角--四元数

#include "std\_msgs/msg/string.hpp"

using namespace std::chrono\_literals;

class TF\_broadcaster : public rclcpp::Node

{

public:

TF\_broadcaster(): Node("TF\_broadcaster")

{

this->declare\_parameter("turtle", "turtle1");

turtle = this->get\_parameter("turtle").as\_string();//乌龟名字动态获取

broadcaster\_ = std::make\_shared<tf2\_ros::TransformBroadcaster>(this);//创建一个广播器

pose\_sub\_ = this->create\_subscription<turtlesim::msg::Pose>("/" + turtle + "/pose", 10,

std::bind(&TF\_broadcaster::do\_pose,this,std::placeholders::\_1));//订阅乌龟1位姿关系

}

private:

std::string turtle;

void do\_pose(const turtlesim::msg::Pose &pose)

{

geometry\_msgs::msg::TransformStamped ts;//获取乌龟1位姿相对world的关系并发布

ts.header.stamp = this->now();

ts.header.frame\_id = "world";

ts.child\_frame\_id = turtle;

ts.transform.translation.x = pose.x;

ts.transform.translation.y = pose.y;

ts.transform.translation.z = 0.0;

tf2::Quaternion qtn;

qtn.setRPY(0, 0, pose.theta);

ts.transform.rotation.x = qtn.x();

ts.transform.rotation.y = qtn.y();

ts.transform.rotation.z = qtn.z();

ts.transform.rotation.w = qtn.w();

broadcaster\_->sendTransform(ts);

}

std::shared\_ptr<tf2\_ros::TransformBroadcaster> broadcaster\_;

rclcpp::Subscription<turtlesim::msg::Pose>::SharedPtr pose\_sub\_;

};

int main(int argc, char \* argv[])

{

rclcpp::init(argc, argv);

rclcpp::spin(std::make\_shared<TF\_broadcaster>());

rclcpp::shutdown();

return 0;

}

\*\*使用ROS2编程 tf\_listener.cpp\*\*

#include <chrono>

#include <memory>

#include "rclcpp/rclcpp.hpp"

#include "tf2\_ros/buffer.h"//缓存器,可以对多个广播器的内容进行存储

#include "tf2\_ros/transform\_listener.h"//创建监听器的类型

#include "geometry\_msgs/msg/twist.hpp"

#include "std\_msgs/msg/string.hpp"

using namespace std::chrono\_literals;

class TF\_listener : public rclcpp::Node

{

public:

TF\_listener(): Node("TF\_listener")

{

//声明参数服务

this->declare\_parameter("father\_frame", "turtle2");

this->declare\_parameter("child\_frame", "turtle1");

father\_frame = this->get\_parameter("father\_frame").as\_string();

child\_frame = this->get\_parameter("child\_frame").as\_string();

buffer\_ = std::make\_unique<tf2\_ros::Buffer>(this->get\_clock());

listener\_ = std::make\_shared<tf2\_ros::TransformListener>(\*buffer\_, this);

timer\_ = this->create\_wall\_timer(1s, std::bind(&TF\_listener::on\_timer, this));

cmd\_pub\_ = this->create\_publisher<geometry\_msgs::msg::Twist>("/" + father\_frame + "/cmd\_vel", 10);

}

private:

void on\_timer()

{

try

{

//实现坐标变换

auto ts = buffer\_->lookupTransform(father\_frame, child\_frame, tf2::TimePointZero);

RCLCPP\_INFO(this->get\_logger(), "坐标转换完成");

RCLCPP\_INFO(this->get\_logger(),

"frame:%s, child\_frame:%s,偏移量(%0.2f, %0.2f, %0.2f)",

ts.header.frame\_id.c\_str(),

ts.child\_frame\_id.c\_str(),

ts.transform.translation.x,

ts.transform.translation.y,

ts.transform.translation.z);

//组织并发布速度指令

geometry\_msgs::msg::Twist twist;

twist.linear.x = 0.5 \* sqrt( pow(ts.transform.translation.x, 2) + pow(ts.transform.translation.y, 2));

twist.angular.z = 1.0 \* atan2(ts.transform.translation.y, ts.transform.translation.x);

cmd\_pub\_->publish(twist);

}

catch(const tf2::LookupException& e)

{

RCLCPP\_INFO(this->get\_logger(), "异常提示：%s", e.what());

}

}

std::string father\_frame;

std::string child\_frame;

std::unique\_ptr<tf2\_ros::Buffer> buffer\_;

std::shared\_ptr<tf2\_ros::TransformListener> listener\_;

rclcpp::TimerBase::SharedPtr timer\_;

rclcpp::Publisher<geometry\_msgs::msg::Twist>::SharedPtr cmd\_pub\_;

};

int main(int argc, char \* argv[])

{

rclcpp::init(argc, argv);

rclcpp::spin(std::make\_shared<TF\_listener>());

rclcpp::shutdown();

return 0;

}

由于题目二涉及许多节点的建立，为了方便，ROS2允许使用launch文件来一次性建立多个节点，实现ROS高效开发。Launch文件通常是Python文件

\*\*使用ROS2编程 turtle\_follows\_launch.py\*\*

from launch import LaunchDescription

from launch\_ros.actions import Node

def generate\_launch\_description():

#启动一个乌龟节点

turtle = Node(package="turtlesim", executable="turtlesim\_node")

#启动spawn节点召唤第二只乌龟

spawn = Node(package="turtle\_follows", executable="spawn", parameters=[{"turtle\_name":"t2"}])

#广播两只乌龟相对world的坐标变换

broadcaster1 = Node(package="turtle\_follows", executable="tf\_broadcaster", name="broad1")

broadcaster2 = Node(package="turtle\_follows", executable="tf\_broadcaster", name="broad2", parameters=[{"turtle":"t2"}])

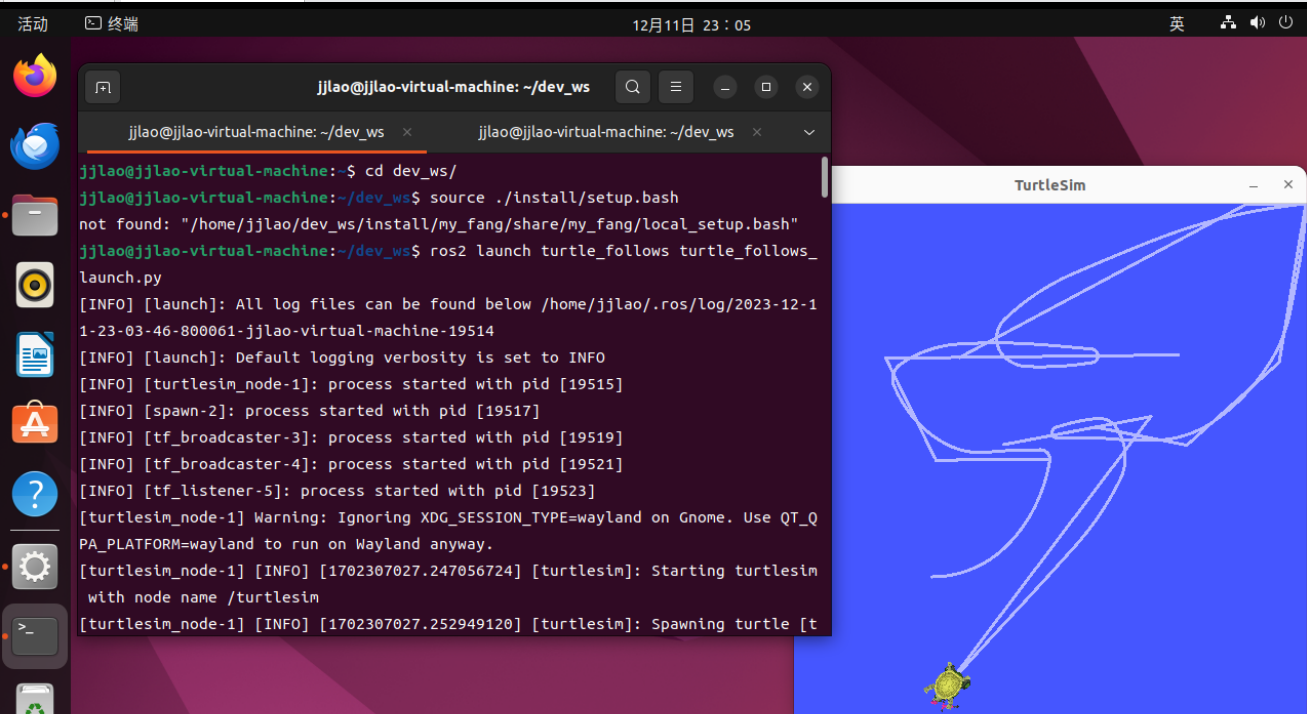
#创建监听节点

listener = Node(package="turtle\_follows", executable="tf\_listener",

parameters=[{"father\_frame":"t2", "child\_frame":"turtle1"}])

return LaunchDescription([turtle ,spawn,broadcaster1,broadcaster2,listener])

最后，关于如何调用launch文件，我们需要进入工作区，调用ros2 launch turtle\_follows turtle\_follows\_launch.py即可同时建立多个节点。



如图所示，乌龟2成功实现对乌龟1的跟随运动。

结论：

通过本次概率机器人三级项目的学习，我组使用ROS2完成题目一和题目二的程序编写我认为我们已经初步理解ROS2的相关开发框架，能进行ROS2相关知识技能的搜索和学习，能尝试进行相关简单程序的编写与学习，具备开发机器人系统最基本的能力。