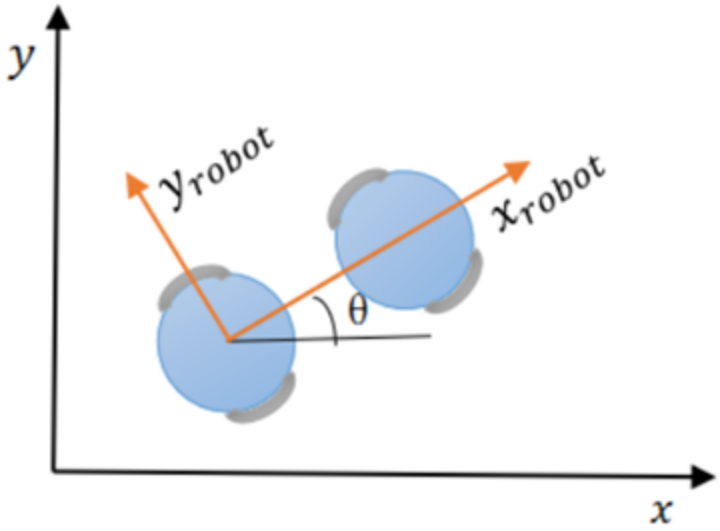
基于T-bot底盘实验指导书---odom话题发布

1.考核目标

1.1 实验背景

在移动机器人领域，里程计（Odometry）是评估机器人位置和运动的重要技术。里程计通过测量轮子的旋转或传感器数据，计算出机器人在环境中的移动轨迹。随着机器人技术的发展，里程计在自主导航、路径规划和环境感知等方面发挥着越来越重要的作用。本实验旨在加深对移动机器人运动学的理解，为后续的机器人控制和应用打下坚实的基础。通过理论与实践相结合的方式，学生将能够掌握移动机器人的基本运动学原理，学会利用ROS开发机器人系统，并培养解决实际问题的能力。

$$x = x + v\Delta t\cos(\theta_t)$$
$$y = y + v\Delta t\sin(\theta_t)$$
$$\theta_t = \theta_t + w\Delta t$$



- **节点运行**
  输入如下指令
  ros2 run teleop_twist_keyboard teleop_twist_keyboard
  启动键盘控制结点，该结点是系统默认自带的，不需要我们去编写这个结点。
- **消息类型**
  单独打开终端，通过查看/cmd_vel，终端中输入ros2 topic info /cmd_vel获得该话题的类型

Type:geometry_msgs/msg/Twis

关于odom话题可以参考odom官方解释

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

1.2 实验目标

- **创建消息发布功能**：制作一个 ROS 节点，发布 `/odom` 消息。
- **验证响应能力**：测试小车在不同速度和方向命令下的响应能力，确保其能够按预期进行移动，记录odom的数据。
- **数据记录与分析**：记录实验过程中小车的运动数据，与实际距离进行比较，以便进行后续分析。

2.实验步骤

2.1创建包

基于上次的消息订阅实验，本实验的工程无需创建，采用之前的my_base即可。

2.2编写发布消息代码

打开my_base.cpp,文件引用头文件bot_serial.h，该头文件中定义了下发指令的结构体和基本类

```
#define SEND_DATA_CHECK   1
#define READ_DATA_CHECK   0
#define FRAME_HEADER      0X7B       //Frame head
#define FRAME_TAIL        0X7D       //Frame tail
#define RECEIVE_DATA_SIZE 24
#define SEND_DATA_SIZE    11
#define PI                                    3.1415926f //PI //圆周率
//The structure in which the lower computer sends data to the ROS
//下位机向ROS发送数据的结构体
typedef struct _RECEIVE_DATA_
{
    uint8_t rx[RECEIVE_DATA_SIZE];
    uint8_t Flag_Stop;
    unsigned char Frame_Header;
    float X_speed;
    float Y_speed;
    float Z_speed;
    float Power_Voltage;
    unsigned char Frame_Tail;
}RECEIVE_DATA;
```

该结构体描述了stm32微控制器上发数据的详细细节，分析可知，X_speed代表车X轴的速度，Y_speed代表车Y轴的速度，Z_speed代表车旋转速度。rx[RECEIVE_DATA_SIZE]表示通过串口上发的数据

数据内容	帧头(固定值 0x7B)		flag_stop		X 轴速度		Y 轴速度		Z 轴速度		X 轴加速度		Y 轴加速度	
数据类型	Uint8		Uint8		short		short		short		short		short	
占用字节	1		1		2		2		2		2		2	
高低位序					MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB
数组号	0		1		2	3	4	5	6	7	8	9	10	11
数据内容	Z 轴加速度		X 轴角速度		Y 轴角速度		Z 轴角速度		电池电压		数据校验位		帧尾(固定值 0x7D)	
数据类型	short		short		short		short		short		Uint8		Uint8	
占用字节	2		2		2		2		2		1		1	
高低位序	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB				
数组号	12	13	14	15	16	17	18	19	20	21	22		23	

表 1-1 上行数据包格式表

```
class turn_on_robot : public rclcpp::Node
{
  public:
    turn_on_robot();  //Constructor //构造函数
    ~turn_on_robot(); //Destructor //析构函数
    void Control();   //Loop control code //循环控制代码
    serial::Serial Stm32_Serial; //Declare a serial object //声明串口对象
  private:
    rclcpp::Time _Now, _Last_Time;  //Time dependent, used for
    float Sampling_Time;
    RECEIVE_DATA Receive_Data;
    SEND_DATA Send_Data;
    bool Get_Sensor_Data_New();
    unsigned char Check_Sum(unsigned char Count_Number,unsigned char mode);
    //check function //校验函数
    short IMU_Trans(uint8_t Data_High,uint8_t Data_Low);
    //IMU data conversion read //IMU数据转化读取
    float Odom_Trans(uint8_t Data_High,uint8_t Data_Low);
    //Odometer data is converted to read //里程计数据转化读取
  };
```

为了建立底盘与ROS系统之间的联系，有必要建立一个类，并在内中定义串口接 `serial::Serial Stm32_SerialStm32_Serial` ,其中 `Get_senosr_Data_New()` 函数与 `Check_Sum()` 函数不属于考核点，老师已经实现了，直接调用即可。学生需要完成类的构造函数 `turn_on_robot()` 和 `Control()` 函数

my_base.c主要由四个函数，请将以下四个函数拷贝到文件中，并实现缺失的代码。

- ***函数1***

```
#include "bot_serial.h"
int main(int argc, char **argv) {
    rclcpp::init(argc, argv);
    turn_on_robot Robot_Control;//Instantiate an object //实例化一个对象
    Robot_Control.Control();
    rclcpp::shutdown();
    return 0;
}
```

- ***函数2***
  可见，我们同时需要在my_base.c文件中需要实现Robot_Control.Control()函数，该函数的模板如下：

```
void turn_on_robot::Control()
{
    while(rclcpp::ok())
    {
        try
        {
            rclcpp::spin_some(this->get_node_base_interface());
        }
        catch (const rclcpp::exceptions::RCLError & e )
        {
            RCLCPP_ERROR(this->get_logger(),"unexpectedly failed whith %s",e.what());
        }
    }
}
```

需要在该函数内完成里程计的计算，并实现话题的发布。

```
void turn_on_robot::Control()
{
  //_Last_Time = ros::Time::now();
    _Last_Time = rclcpp::Node::now();
    while(rclcpp::ok())
    {
        try
        {
            _Now = rclcpp::Node::now();
            Sampling_Time = (_Now - _Last_Time).seconds();  //Retrieves
            if (true == Get_Sensor_Data_New()) //The serial port reads and
            {
                //填写公式

                Publish_Odom();      //Pub the speedometer topic //发布里程计话题
                _Last_Time = _Now;
            }
            rclcpp::spin_some(this->get_node_base_interface());   //The loop
        }
        catch (const rclcpp::exceptions::RCLError & e )
        {
            RCLCPP_ERROR(this->get_logger(),"unexpectedly failed whith %s",e.what());
        }
    }
}
```

- ***函数4，odom话题发布***

```
void turn_on_robot::Publish_Odom()
{
    //Convert the Z-axis rotation Angle into a quaternion for expression
    //把Z轴转角转换为四元数进行表达
    tf2::Quaternion q;
    q.setRPY(0,0,Robot_Pos.Z);
    geometry_msgs::msg::Quaternion odom_quat=tf2::toMsg(q);

    nav_msgs::msg::Odometry odom; //Instance the odometer topic data //实例化里程计话题数据
    odom.header.stamp = rclcpp::Node::now(); ;
    odom.header.frame_id = odom_frame_id; // Odometer TF parent coordinates //里程计TF父坐标
    odom.pose.pose.position.x = Robot_Pos.X; //Position //位置
    odom.pose.pose.position.y = Robot_Pos.Y;
    odom.pose.pose.position.z = Robot_Pos.Z;
    odom.pose.pose.orientation = odom_quat; //Posture, Quaternion converted by Z-axis rotation ,

    odom.child_frame_id = robot_frame_id; // Odometer TF subcoordinates //里程计TF子坐标
    odom.twist.twist.linear.x =  Robot_Vel.X; //Speed in the X direction //X方向速度
    odom.twist.twist.linear.y =  Robot_Vel.Y; //Speed in the Y direction //Y方向速度
    odom.twist.twist.angular.z = Robot_Vel.Z; //Angular velocity around the Z axis //绕Z轴角速度

    //There are two types of this matrix, which are used when the robot is at rest and when it i
    //这个矩阵有两种，分别在机器人静止和运动的时候使用。扩展卡尔曼滤波官方提供的2个矩阵，用于robot_po
    if(Robot_Vel.X== 0&&Robot_Vel.Y== 0&&Robot_Vel.Z== 0)
      //If the velocity is zero, it means that the error of the encoder will be relatively smal
      //如果velocity是零，说明编码器的误差会比较小，认为编码器数据更可靠
      memcpy(&odom.pose.covariance, odom_pose_covariance2, sizeof(odom_pose_covariance2)),
      memcpy(&odom.twist.covariance, odom_twist_covariance2, sizeof(odom_twist_covariance2));
    else
      //If the velocity of the trolley is non-zero, considering the sliding error that may be br
      //如果小车velocity非零，考虑到运动中编码器可能带来的滑动误差，认为imu的数据更可靠
      memcpy(&odom.pose.covariance, odom_pose_covariance, sizeof(odom_pose_covariance)),
      memcpy(&odom.twist.covariance, odom_twist_covariance, sizeof(odom_twist_covariance));
    odom_publisher->publish(odom); //Pub odometer topic //发布里程计话题
}
```

2.3修改package.xml

进入ros2_ws/src/my_base目录并打开package.xml，按照之前教程要求填写description ， maintainer和license.如果你并不想开源你的代码，可以忽略此步。

```
    <description>TODO: Package description</description>
    <maintainer email="nxrobo@todo.todo">nxrobo</maintainer>
    <license>TODO: License declaration</license>
```

在编译工具依赖ament_cmake后

```
  <buildtool_depend>ament_cmake</buildtool_depend>
```

添加下列依赖项：

```
    <depend>rclcpp</depend>
    <depend>std_msgs</depend>
```

改写完毕后注意记得保存文档!

2.4修改CmakeLists.txt

打开 CMakeLists.txt ，在 find_package(ament_cmake REQUIRED) 下添加两行：

```
find_package(rclcpp REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(tf2_geometry_msgs REQUIRED)
find_package(serial REQUIRED)

add_executable(my_base src/my_base.cpp)
ament_target_dependencies(my_base rclcpp serial tf2_geometry_msgs)
```

然后在添加可执行文件需要编译的源文件，并命名为 talker或者其他你认为合适的名字

3.编译运行

打开终端，执行如下命令：
执行colcon build
执行 source install/setup.bash
执行 ros2 run my_base my_base

4.测试验证

- 控制底盘运动一段距离，同时观察odom输出值变化；
- 采用米尺测量小车X轴方向和Y轴方向的位移，和odom对比，分析两者的区别和产生区别的原理